10/531287

FIG.1

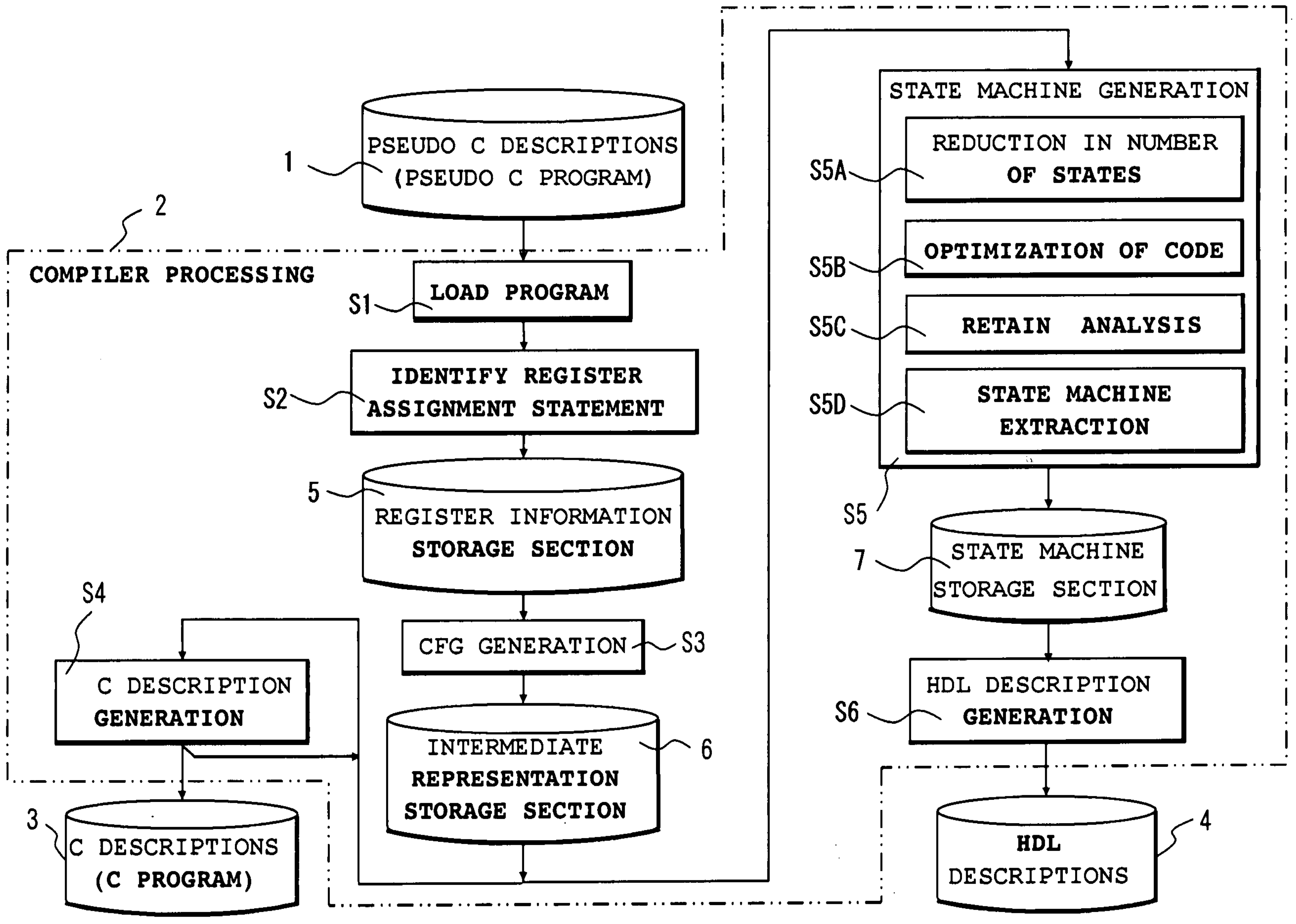
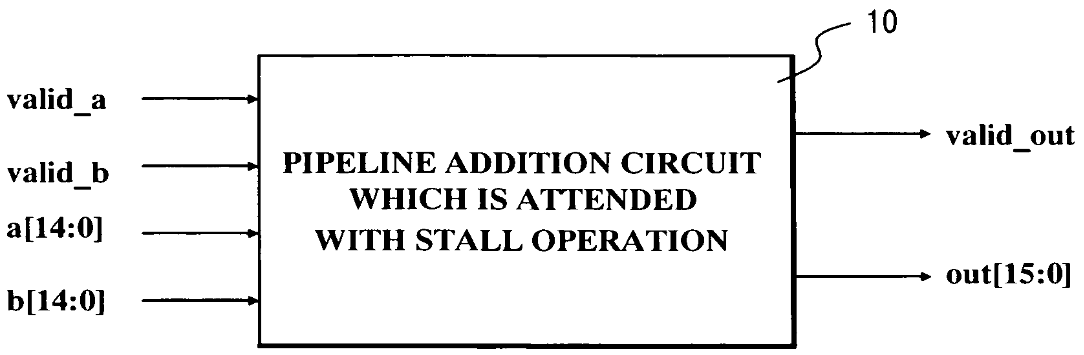
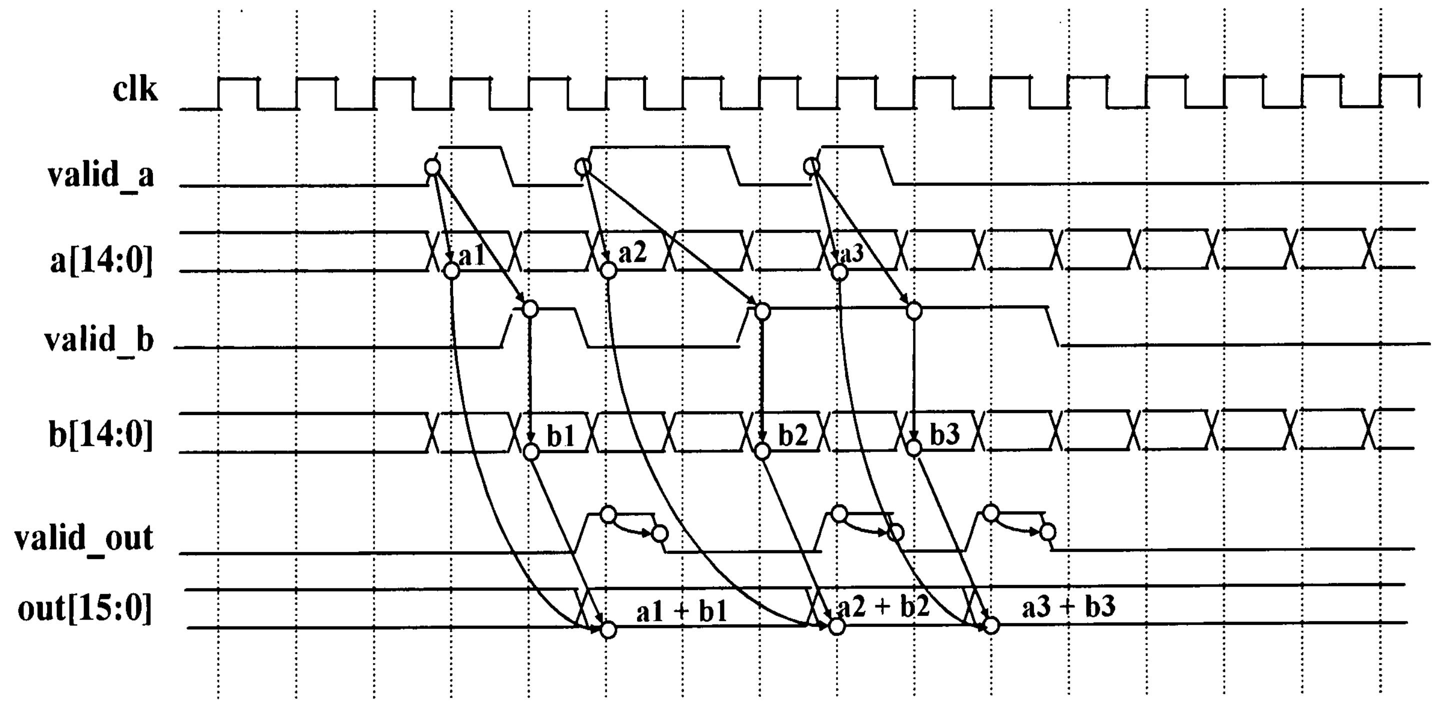

PSEUDO C DESCRIPTIONS (PSEUDO C PROGRAM)
1
2
COMPILER PROCESSING
LOAD PROGRAM
S1
IDENTIFY REGISTER ASSIGNMENT STATEMENT
S2
5
REGISTER INFORMATION STORAGE SECTION
CFG GENERATION
S3
S4
C DESCRIPTION GENERATION
INTERMEDIATE REPRESENTATION STORAGE SECTION
6
3
C DESCRIPTIONS (C PROGRAM)
STATE MACHINE GENERATION
S5A
REDUCTION IN NUMBER OF STATES
S5B
OPTIMIZATION OF CODE
S5C
RETAIN ANALYSIS
S5D
STATE MACHINE EXTRACTION
S5
7
STATE MACHINE STORAGE SECTION
S6
HDL DESCRIPTION GENERATION
4
HDL DESCRIPTIONS

FIG.2


10
valid_a
valid_b
a[14:0]
b[14:0]
PIPELINE ADDITION CIRCUIT WHICH IS ATTENDED WITH STALL OPERATION
valid_out
out[15:0]

10/531287

FIG.3



out[15:0] = a[14:0] + b[14:0]

10/531287

FIG.4

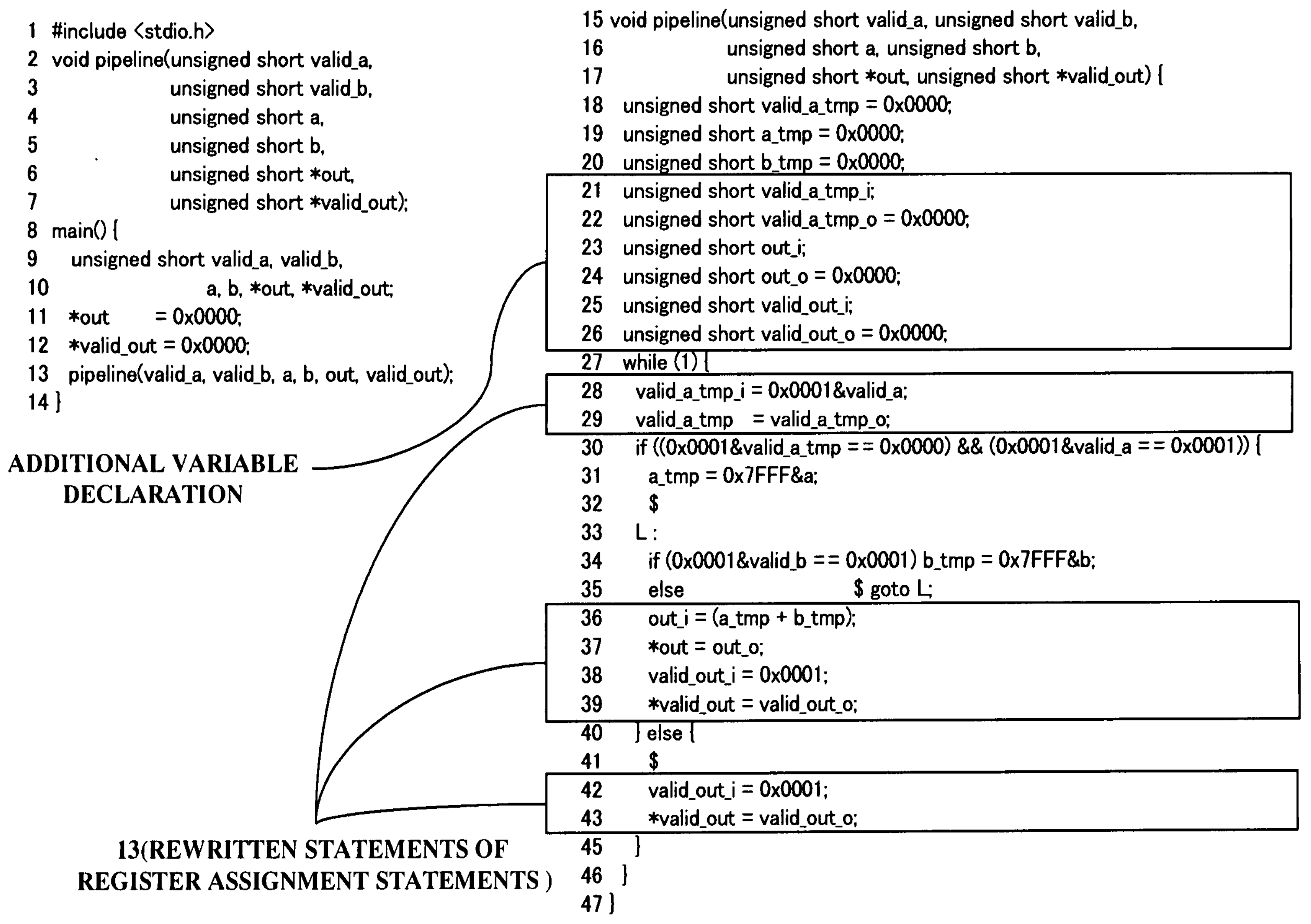
```
#include <stdio.h>
void pipeline(unsigned short valid_a,
              unsigned short valid_b,
              unsigned short a,
              unsigned short b,
              unsigned short *out,
              unsigned short *valid_out);
main() {
  unsigned short valid_a, valid_b,
                 a, b, *out, *valid_out;
  *out       = 0x0000;
  *valid_out = 0x0000;
  pipeline(valid_a, valid_b, a, b, out, valid_out);
}
```

```
void pipeline(unsigned short valid_a, unsigned short valid_b,
              unsigned short a, unsigned short b,
              unsigned short *out, unsigned short *valid_out) {
 unsigned short valid_a_tmp = 0x0000;
 unsigned short a_tmp = 0x0000;
 unsigned short b_tmp = 0x0000;
 while (1) {
  valid_a_tmp = $ 0x0001&valid_a;
  if ((0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)) {
   a_tmp = 0x7FFF&a;
   $
  L :
   if (0x0001&valid_b == 0x0001) b_tmp = 0x7FFF&b;
   else                 $ goto L;
   *out = $ (a_tmp + b_tmp);
   *valid_out = $ 0x0001;
  } else {
   $
   *valid_out = $ 0x0000;
  }
 }
}
```

11 (CIRCUIT OPERATION DESCRIPTION PART)

10/531287

FIG.5

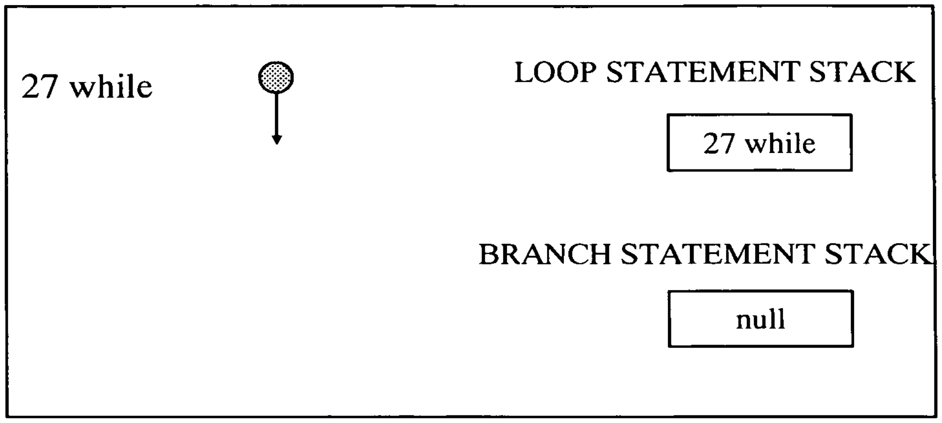


```
1  #include <stdio.h>
2  void pipeline(unsigned short valid_a,
3                unsigned short valid_b,
4                unsigned short a,
5                unsigned short b,
6                unsigned short *out,
7                unsigned short *valid_out);
8  main() {
9    unsigned short valid_a, valid_b,
10                  a, b, *out, *valid_out;
11   *out       = 0x0000;
12   *valid_out = 0x0000;
13   pipeline(valid_a, valid_b, a, b, out, valid_out);
14 }
```

```
15 void pipeline(unsigned short valid_a, unsigned short valid_b,
16              unsigned short a, unsigned short b,
17              unsigned short *out, unsigned short *valid_out) {
18  unsigned short valid_a_tmp = 0x0000;
19  unsigned short a_tmp = 0x0000;
20  unsigned short b_tmp = 0x0000;
21  unsigned short valid_a_tmp_i;
22  unsigned short valid_a_tmp_o = 0x0000;
23  unsigned short out_i;
24  unsigned short out_o = 0x0000;
25  unsigned short valid_out_i;
26  unsigned short valid_out_o = 0x0000;
27  while (1) {
28   valid_a_tmp_i = 0x0001&valid_a;
29   valid_a_tmp   = valid_a_tmp_o;
30   if ((0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)) {
31    a_tmp = 0x7FFF&a;
32    $
33   L :
34    if (0x0001&valid_b == 0x0001) b_tmp = 0x7FFF&b;
35    else                  $ goto L;
36    out_i = (a_tmp + b_tmp);
37    *out = out_o;
38    valid_out_i = 0x0001;
39    *valid_out = valid_out_o;
40   } else {
41    $
42    valid_out_i = 0x0001;
43    *valid_out = valid_out_o;
45   }
46  }
47 }
```

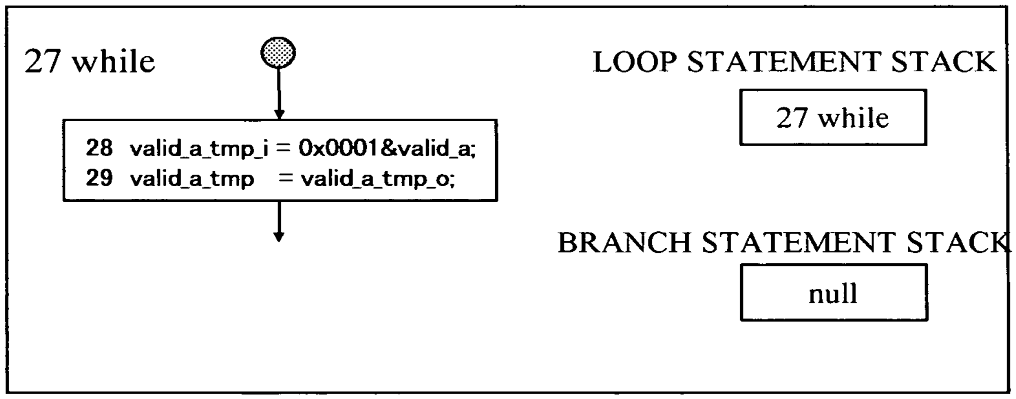
ADDITIONAL VARIABLE DECLARATION
13(REWRITTEN STATEMENTS OF REGISTER ASSIGNMENT STATEMENTS)

FIG.6



27 while
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
null

FIG.7



27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
null

FIG. 8

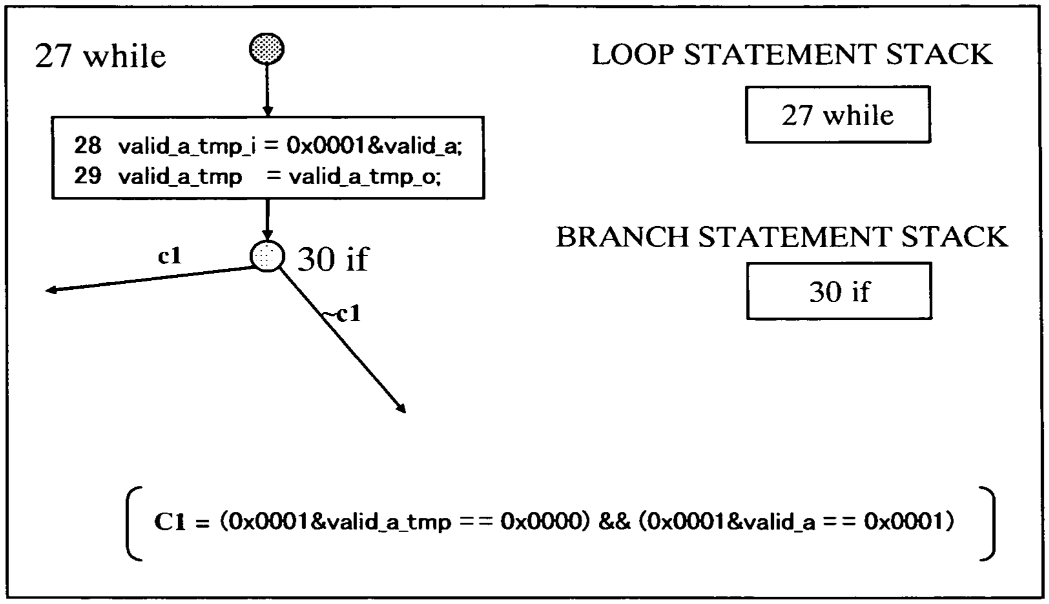

27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
30 if
~c1
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

FIG. 9

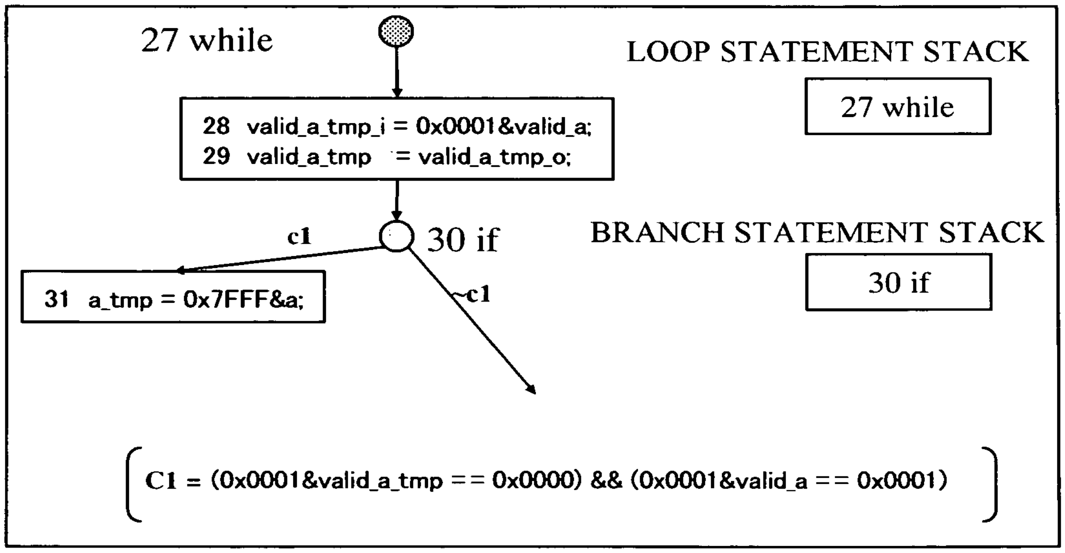
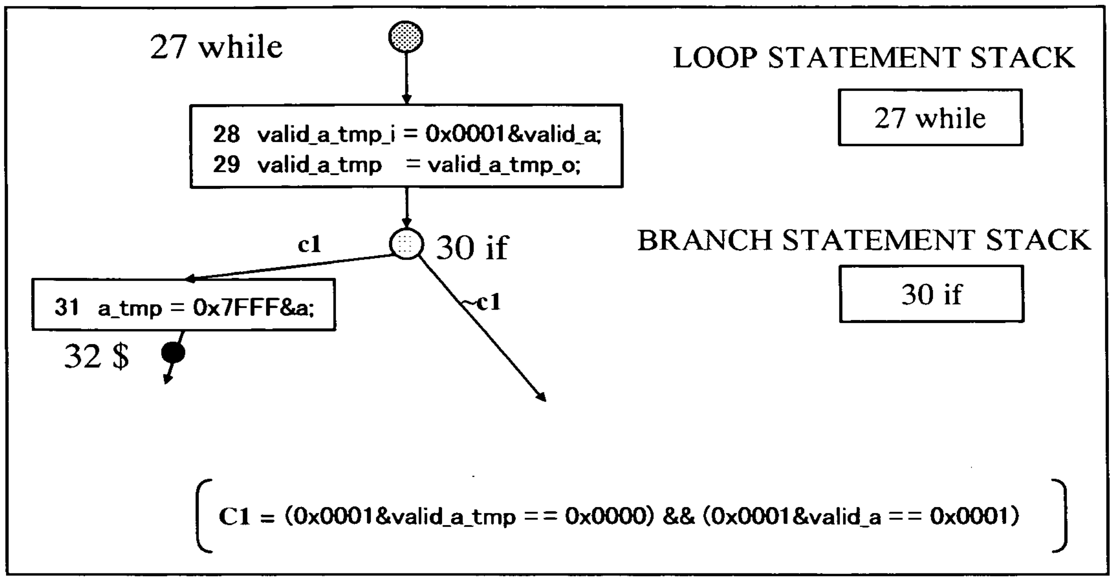
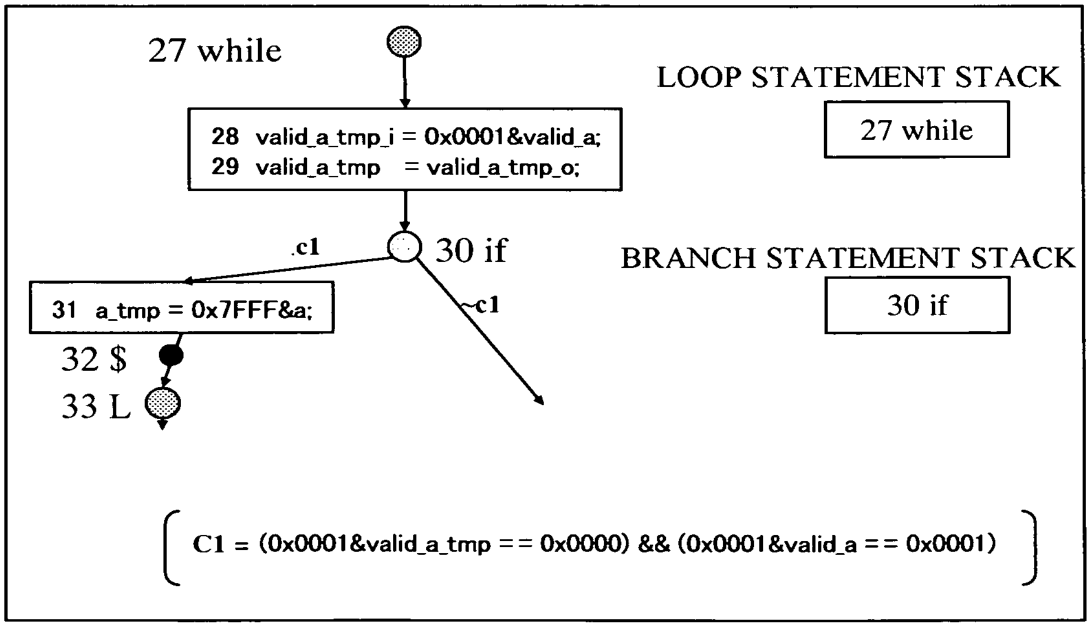

27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
30 if
~c1
31 a_tmp = 0x7FFF&a;
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

FIG.10



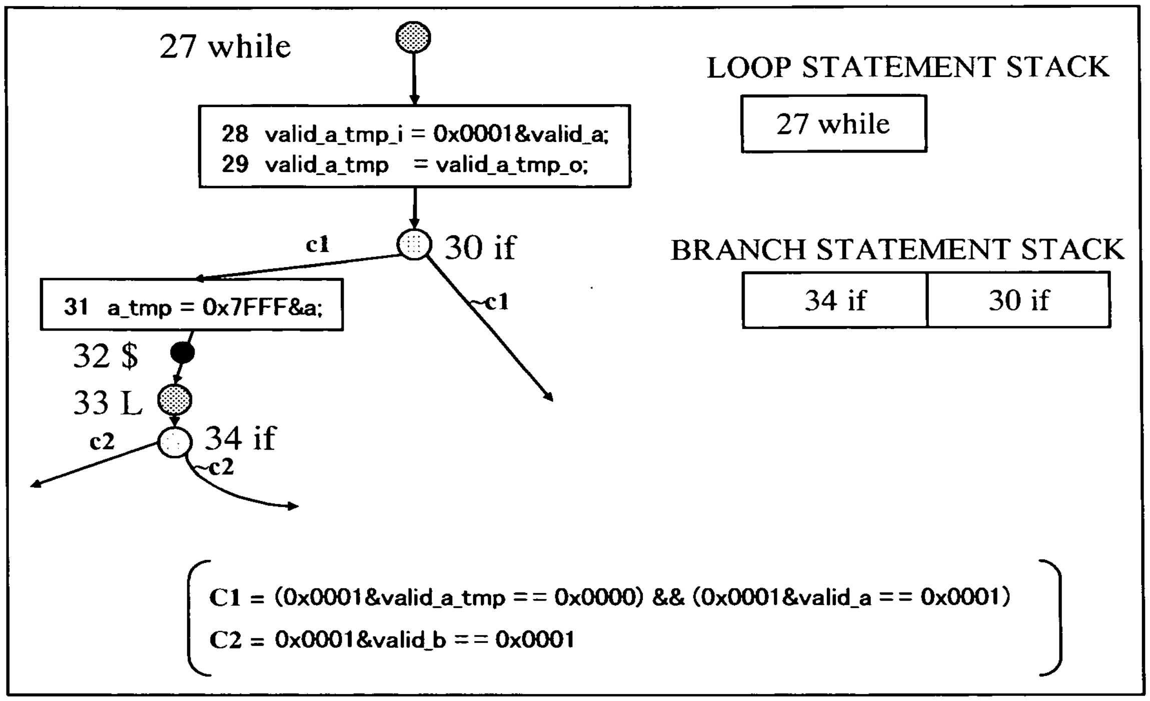
27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
30 if
c1
~c1
31 a_tmp = 0x7FFF&a;
32 $
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

FIG.11



27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
30 if
c1
~c1
31 a_tmp = 0x7FFF&a;
32 $
33 L
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

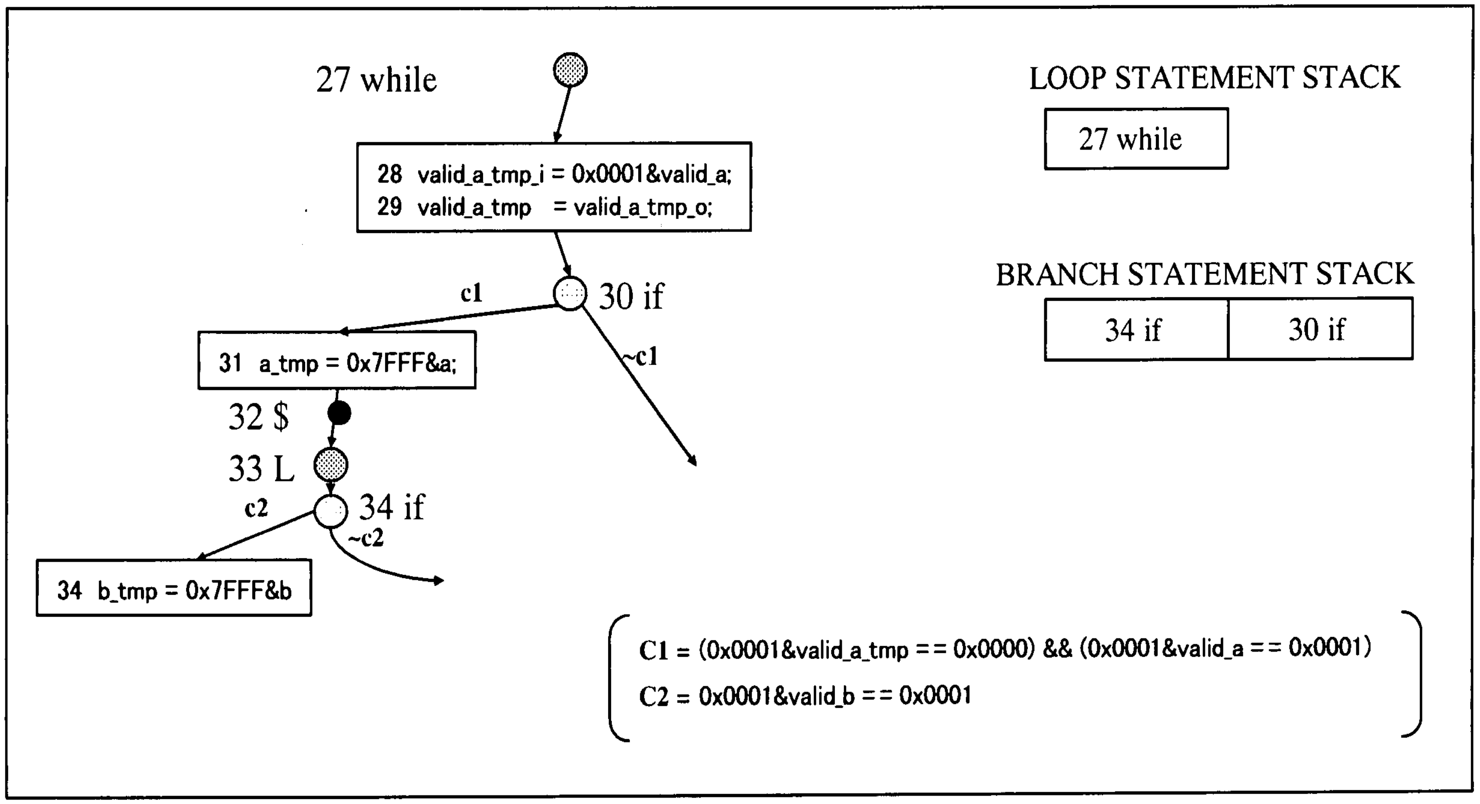
FIG. 12



27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
30 if
~c1
31 a_tmp = 0x7FFF&a;
32 $
33 L
c2
34 if
~c2
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
34 if
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

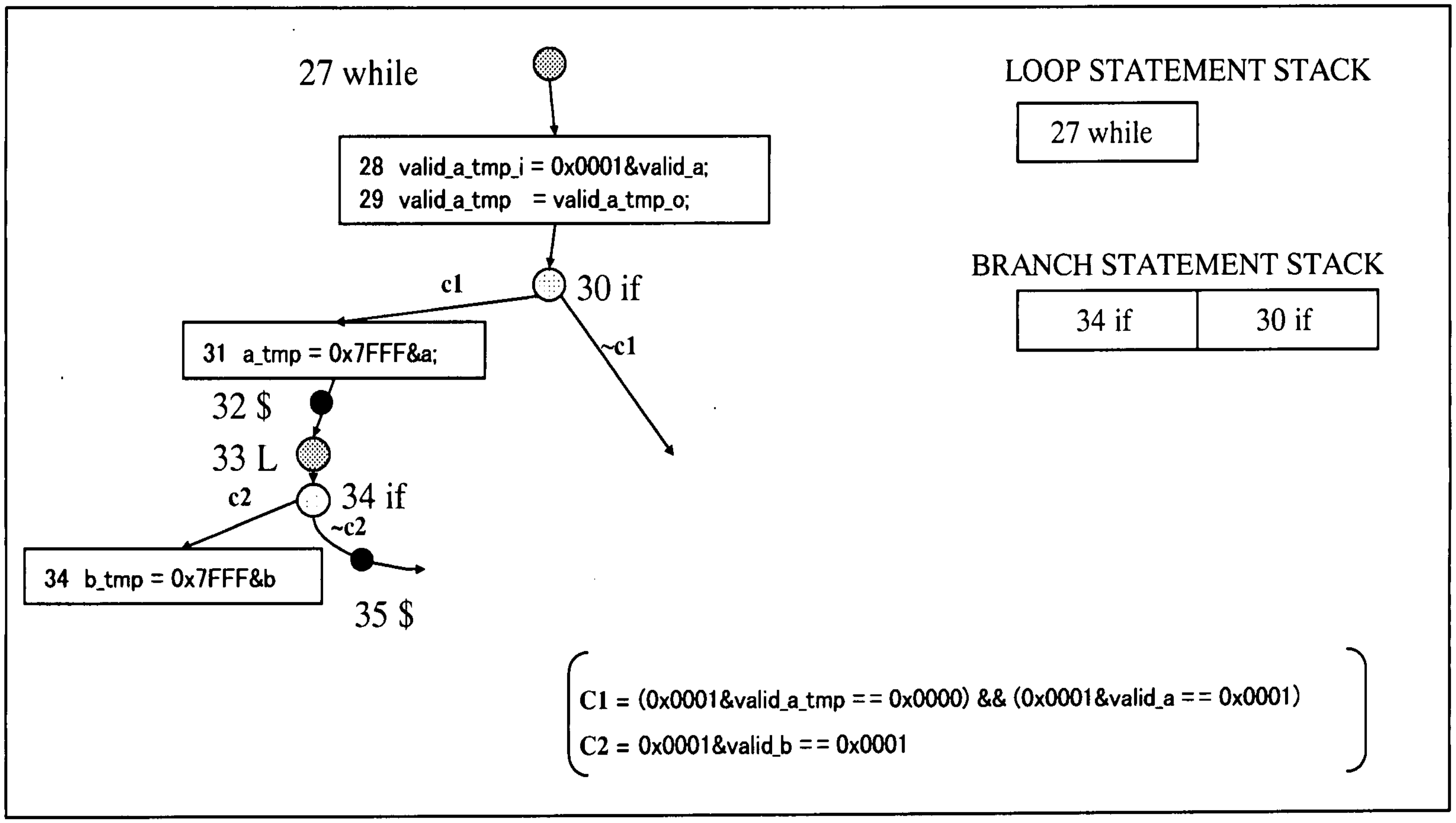
10/531287

FIG.13



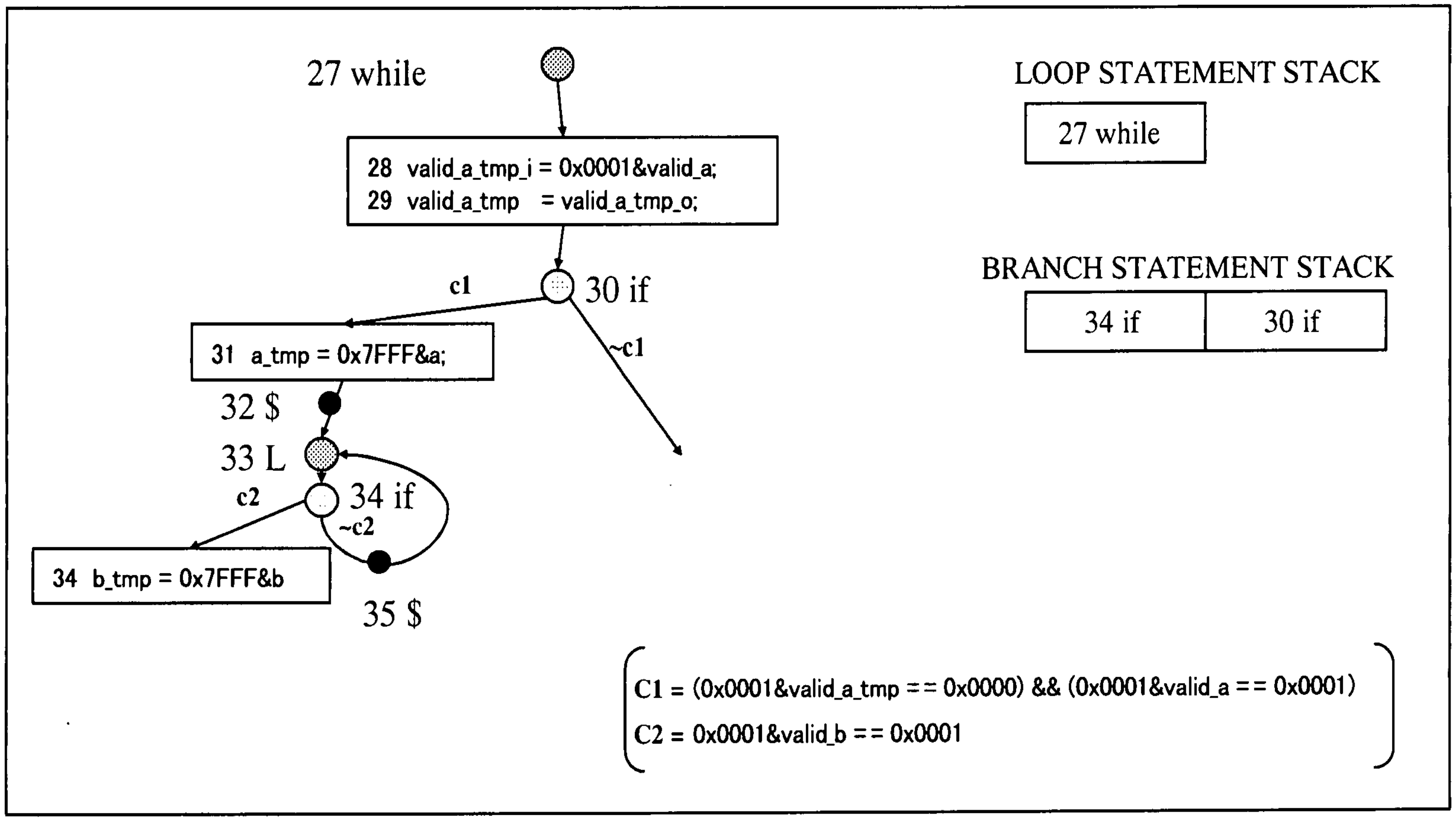
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

FIG.14


27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
30 if
c1
~c1
31 a_tmp = 0x7FFF&a;
32 $
33 L
34 if
c2
~c2
34 b_tmp = 0x7FFF&b
35 $
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
34 if
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

10/531287

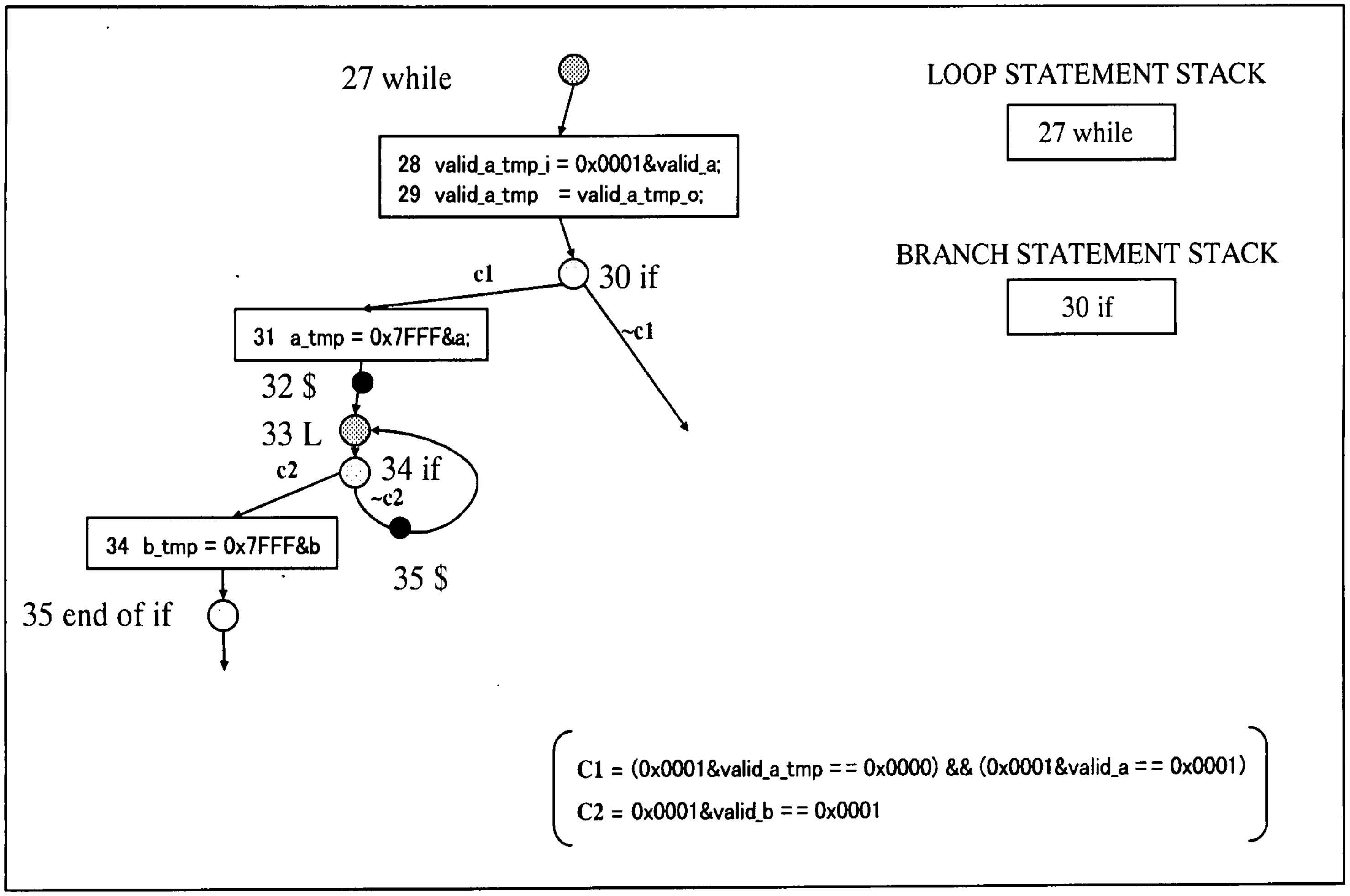
FIG.15



27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
30 if
c1
~c1
31 a_tmp = 0x7FFF&a;
32 $
33 L
34 if
c2
~c2
34 b_tmp = 0x7FFF&b
35 $
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
34 if
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

10/531287

FIG.16


27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
30 if
c1
~c1
31 a_tmp = 0x7FFF&a;
32 $
33 L
34 if
c2
~c2
34 b_tmp = 0x7FFF&b
35 $
35 end of if
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

10/531287

FIG.17

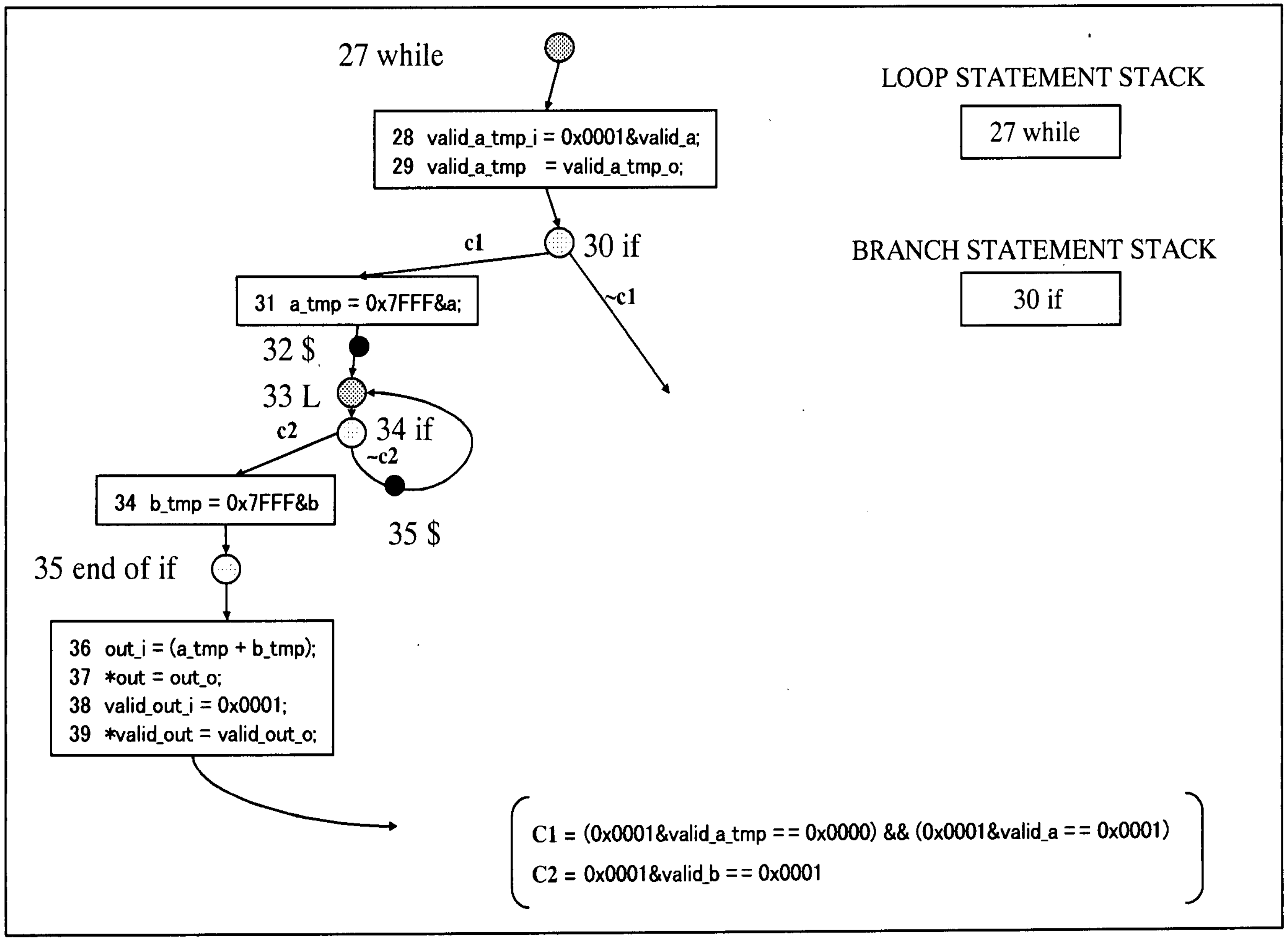
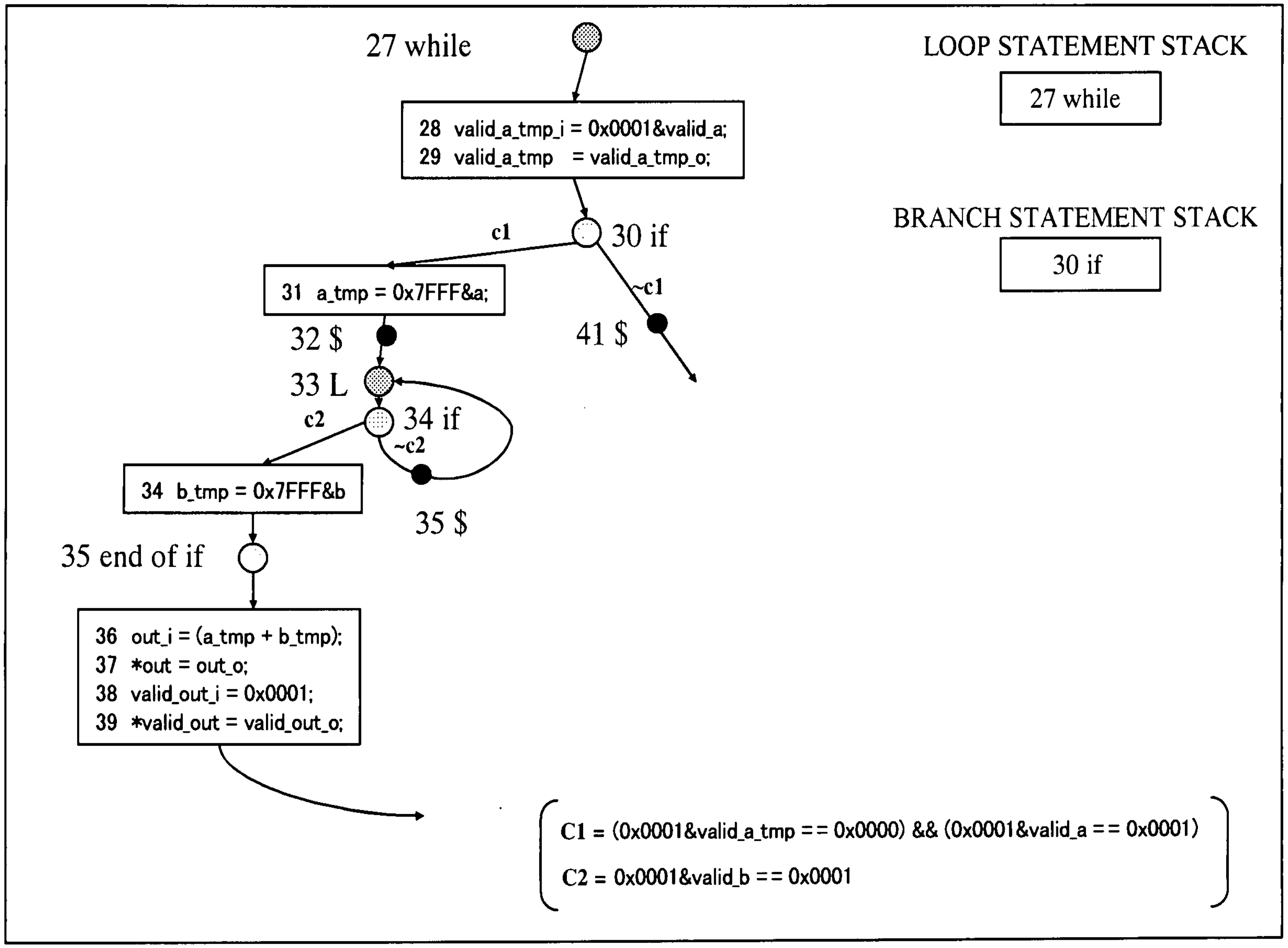

27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
30 if
c1
~c1
31 a_tmp = 0x7FFF&a;
32 $
33 L
34 if
c2
~c2
34 b_tmp = 0x7FFF&b
35 $
35 end of if
36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

FIG.18



```
28  valid_a_tmp_i = 0x0001&valid_a;
29  valid_a_tmp   = valid_a_tmp_o;
31  a_tmp = 0x7FFF&a;
34  b_tmp = 0x7FFF&b
36  out_i = (a_tmp + b_tmp);
37  *out = out_o;
38  valid_out_i = 0x0001;
39  *valid_out = valid_out_o;
```

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)

C2 = 0x0001&valid_b == 0x0001

10/531287

FIG.19

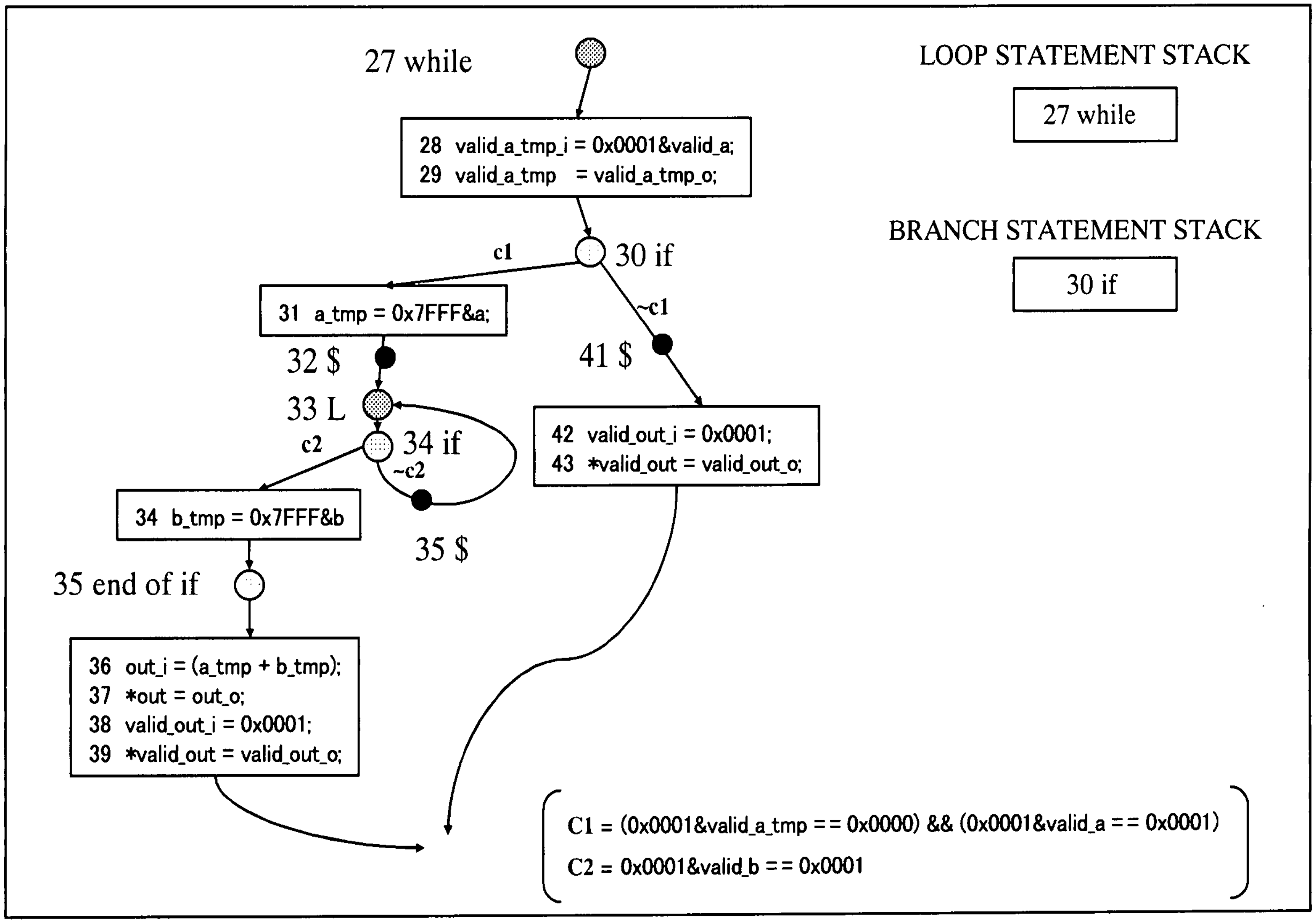
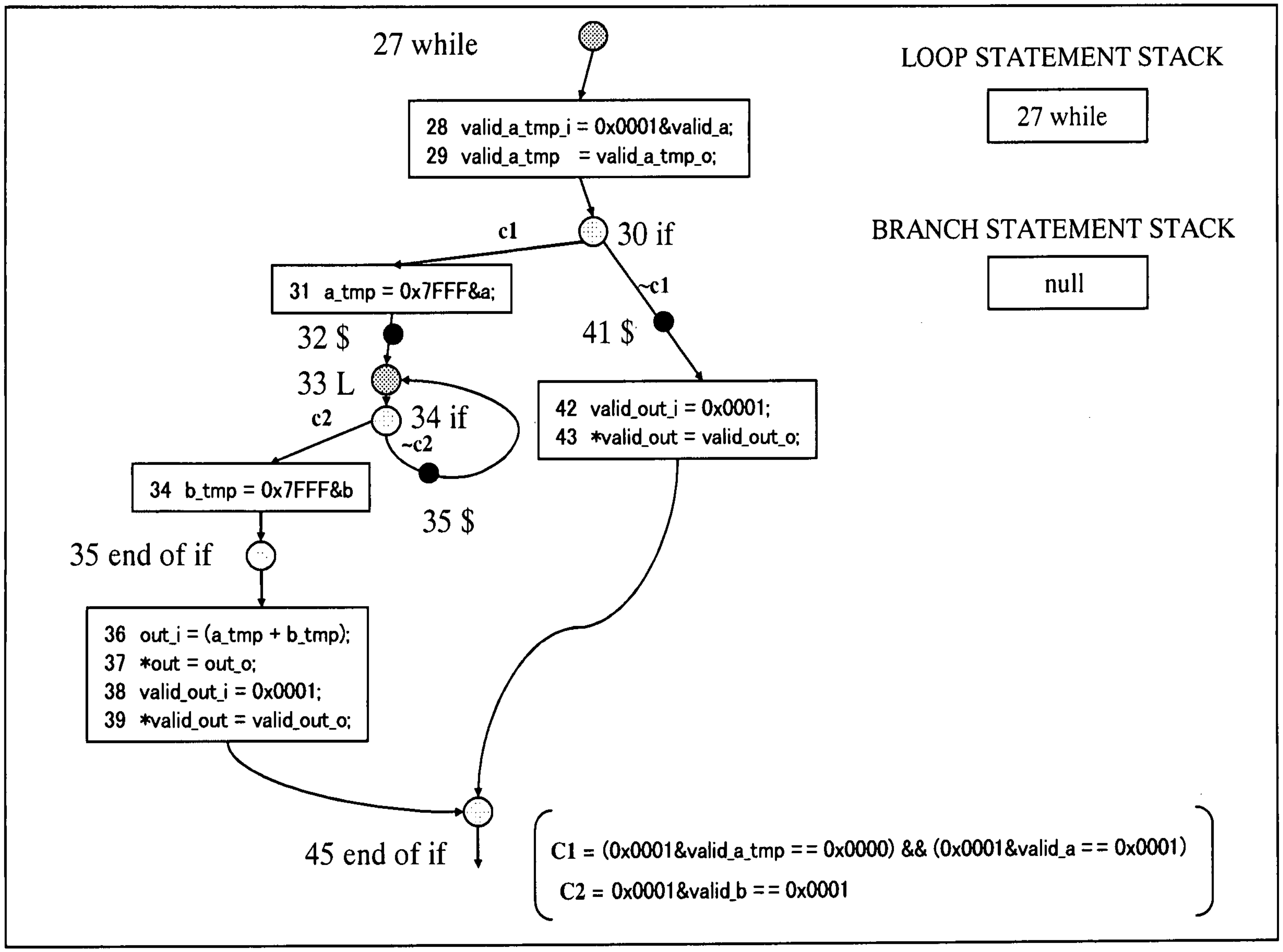

27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
30 if
c1
~c1
31 a_tmp = 0x7FFF&a;
32 $
33 L
34 if
c2
~c2
34 b_tmp = 0x7FFF&b
35 $
35 end of if
36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
41 $
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
30 if
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

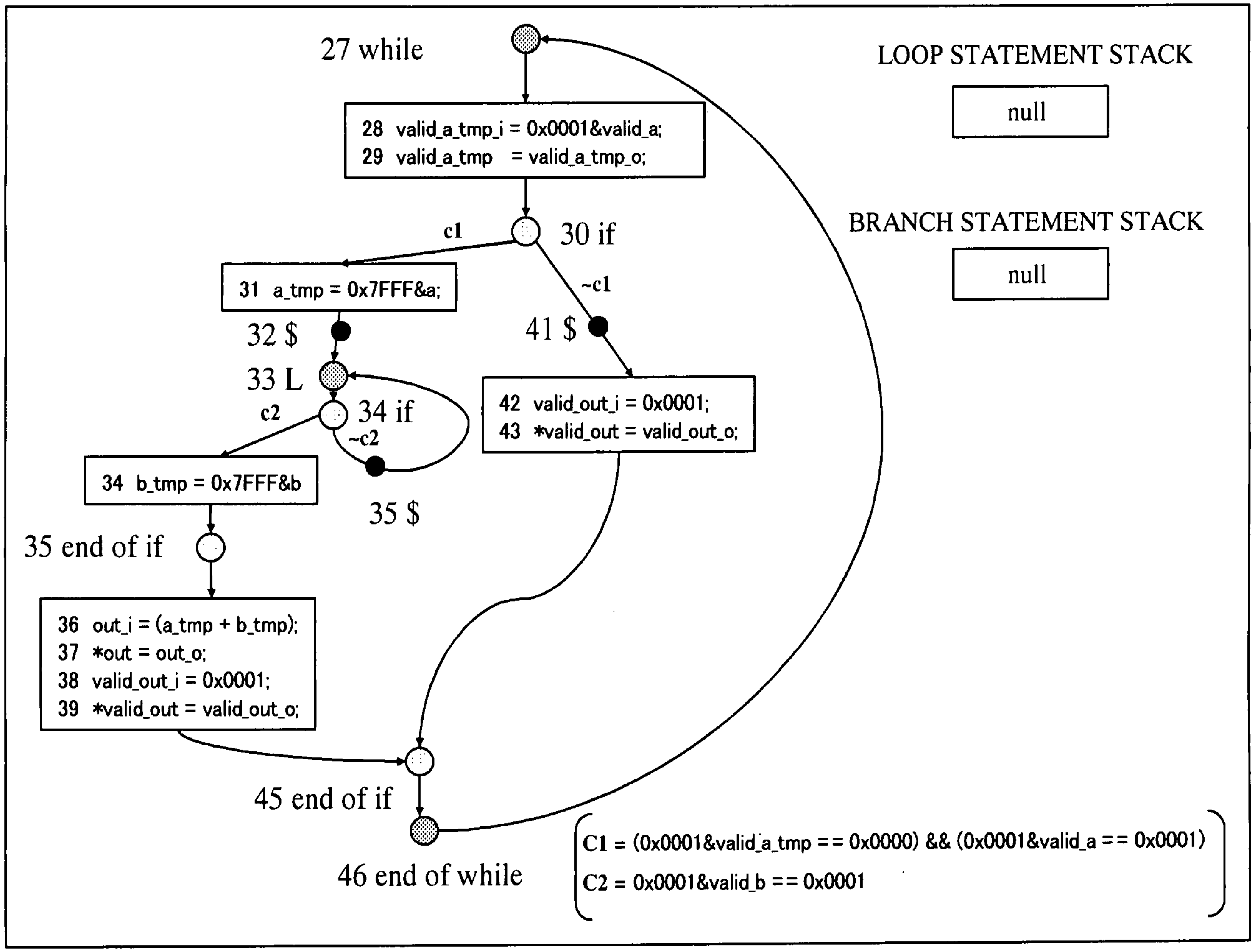
FIG. 20



27 while
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
30 if
c1
~c1
31 a_tmp = 0x7FFF&a;
32 $
33 L
34 if
c2
~c2
34 b_tmp = 0x7FFF&b
35 $
35 end of if
36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
41 $
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
45 end of if
LOOP STATEMENT STACK
27 while
BRANCH STATEMENT STACK
null
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

10/531287

FIG.21



```
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp   = valid_a_tmp_o;
```

```
31 a_tmp = 0x7FFF&a;
```

```
34 b_tmp = 0x7FFF&b
```

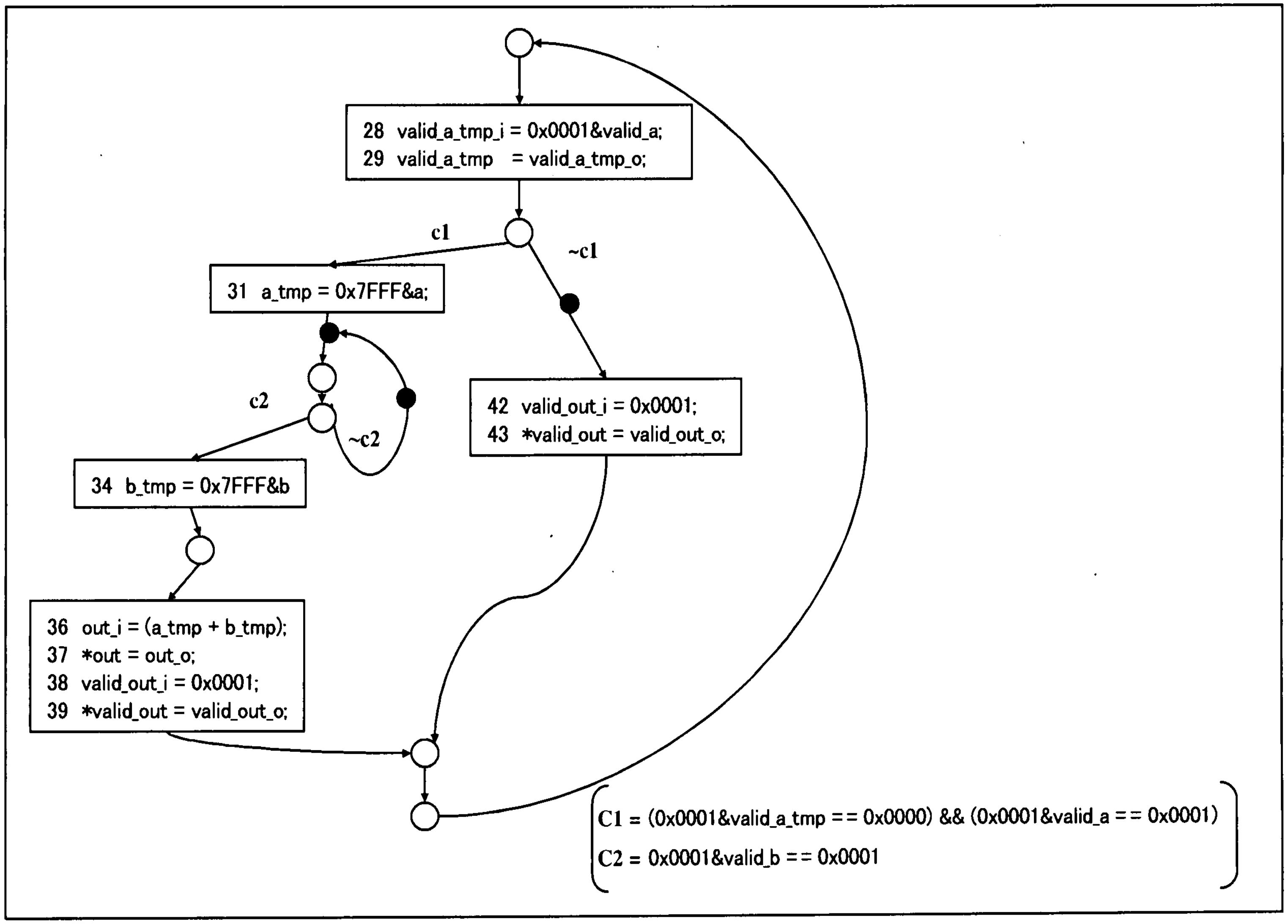
```
36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
```

```
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
```

C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

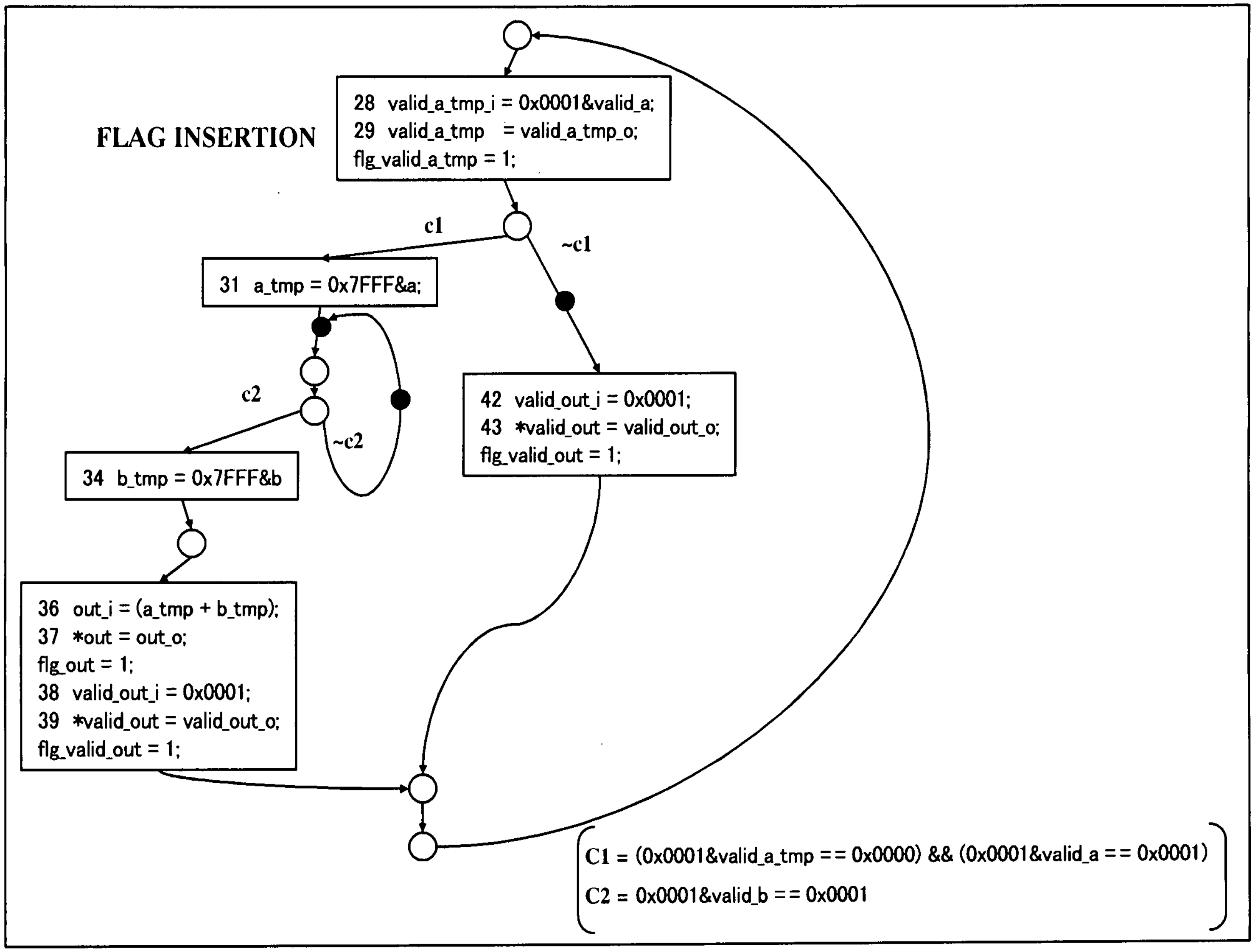
10/531287

FIG.22


28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
~c1
31 a_tmp = 0x7FFF&a;
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
c2
~c2
34 b_tmp = 0x7FFF&b
36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

10/531287

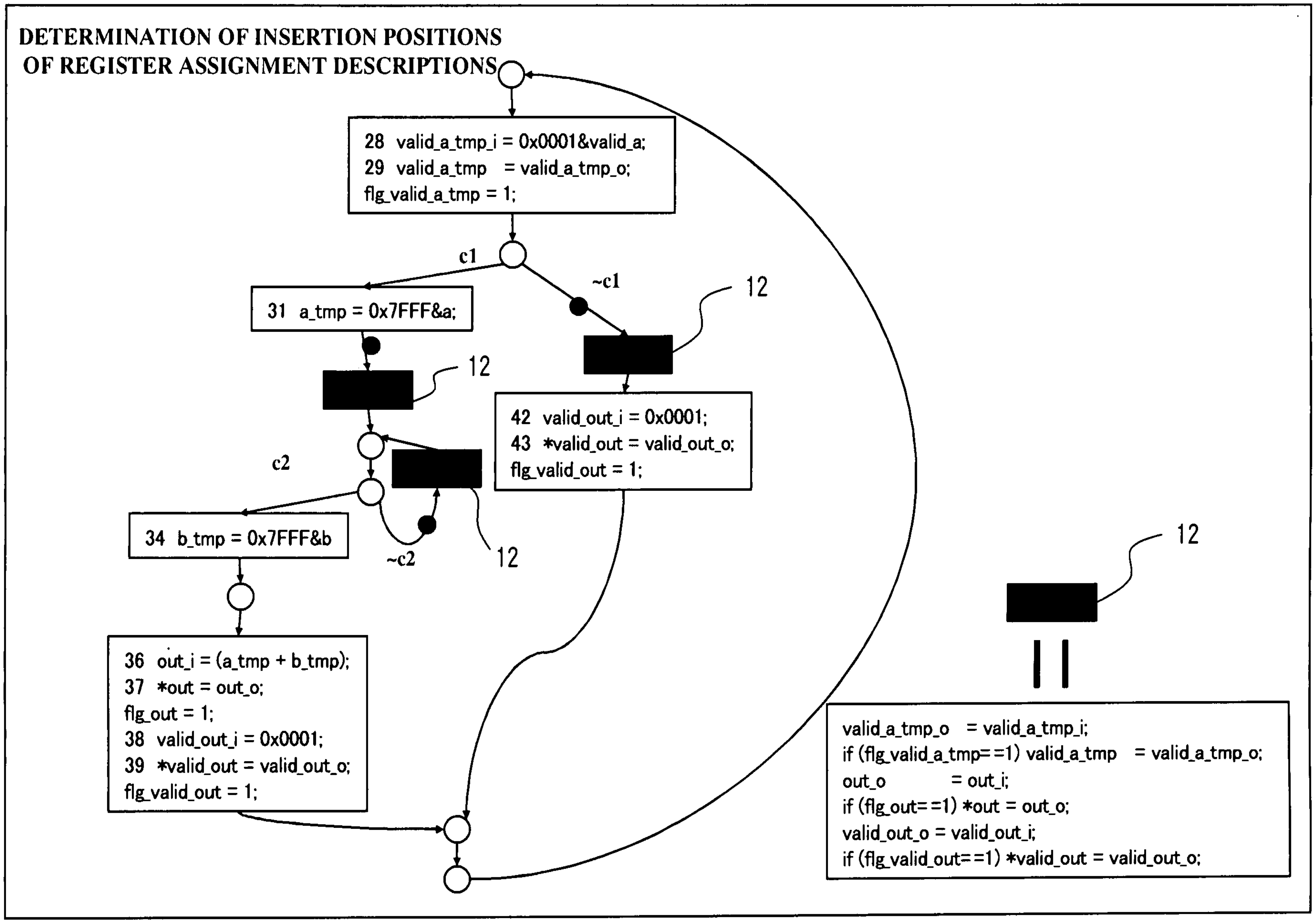
FIG.23



FLAG INSERTION
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
flg_valid_a_tmp = 1;
c1
~c1
31 a_tmp = 0x7FFF&a;
c2
~c2
34 b_tmp = 0x7FFF&b
36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
flg_out = 1;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
flg_valid_out = 1;
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
flg_valid_out = 1;
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

10/531287

FIG.24


DETERMINATION OF INSERTION POSITIONS
OF REGISTER ASSIGNMENT DESCRIPTIONS
c1
~c1
c2
~c2
12
12
12
12

```
28  valid_a_tmp_i = 0x0001&valid_a;
29  valid_a_tmp   = valid_a_tmp_o;
flg_valid_a_tmp = 1;
```

```
31  a_tmp = 0x7FFF&a;
```

```
42  valid_out_i = 0x0001;
43  *valid_out = valid_out_o;
flg_valid_out = 1;
```

```
34  b_tmp = 0x7FFF&b
```

```
36  out_i = (a_tmp + b_tmp);
37  *out = out_o;
flg_out = 1;
38  valid_out_i = 0x0001;
39  *valid_out = valid_out_o;
flg_valid_out = 1;
```

```
valid_a_tmp_o   = valid_a_tmp_i;
if (flg_valid_a_tmp==1) valid_a_tmp   = valid_a_tmp_o;
out_o           = out_i;
if (flg_out==1) *out = out_o;
valid_out_o = valid_out_i;
if (flg_valid_out==1) *valid_out = valid_out_o;
```

10/531287

FIG.25

```
1  #include <stdio.h>
2  void pipeline(unsigned short valid_a,
3                 unsigned short valid_b,
4                 unsigned short a,
5                 unsigned short b,
6                 unsigned short *out,
7                 unsigned short *valid_out);
8  main() {
9   unsigned short valid_a, valid_b,
10                   a, b, *out, *valid_out;
11   *out       = 0x0000;
12   *valid_out = 0x0000;
13   pipeline(valid_a, valid_b, a, b, out, valid_out);
14 }
```

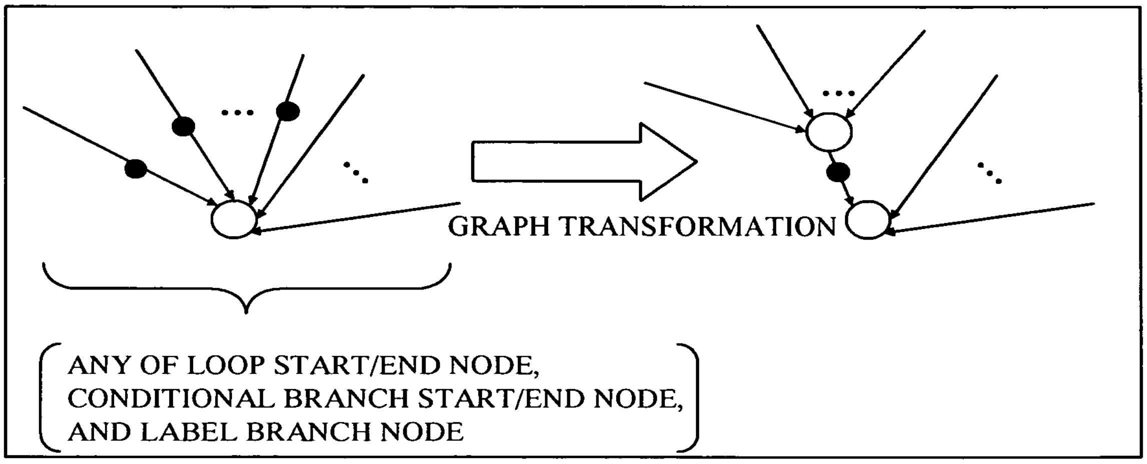
```
15 void pipeline(unsigned short valid_a, unsigned short valid_b,
16                unsigned short a, unsigned short b,
17                unsigned short *out, unsigned short *valid_out) {
18   unsigned short valid_a_tmp      = 0x0000;
19   unsigned short a_tmp            = 0x0000;
20   unsigned short b_tmp            = 0x0000;
     /* Added variables */
21   unsigned short valid_a_tmp_i;
22   unsigned short valid_a_tmp_o    = 0x0000;
23   unsigned short valid_out_i;
24   unsigned short valid_out_o   = 0x0000;
25   unsigned short out_i;
26   unsigned short out_o            = 0x0000;
27   unsigned short flg_valid_a_tmp = 0x0000;
28   unsigned short flg_valid_out = 0x0000;
29   unsigned short flg_out          = 0x0000;
```

10/531287

FIG.26

```
while (1) {
  /* valid_a_tmp = $ valid_a; */
  valid_a_tmp_i = 0x0001&valid_a;        /* Refined */
  valid_a_tmp   = valid_a_tmp_o;         /* Refined */
  flg_valid_a_tmp = 1;
  if ((0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)) {
    a_tmp = 0x7FFF&a;
    /* $ */
    /* BEGIN : Register Assignment */
    valid_a_tmp_o   = valid_a_tmp_i;
    if (flg_value_a_tmp == 1) valid_a_tmp     = valid_a_tmp_o;
    out_o           = out_i;
    if (flg_out==1) *out = out_o;
    valid_out_o = valid_out_i;
    if (flg_valid_out==1) *valid_out = valid_out_o;
     /* END   : Register Assignment */
  L :
```

FIG.28


GRAPH TRANSFORMATION
ANY OF LOOP START/END NODE,
CONDITIONAL BRANCH START/END NODE,
AND LABEL BRANCH NODE

10/531287

FIG.27

```
if (0x0001&valid_b == 0x0001) b_tmp = 0x7FFF&b;
else {
  /* $ */
  /* BEGIN : Register Assignment */
  valid_a_tmp_o   = valid_a_tmp_i;
  valid_a_tmp     = valid_a_tmp_o;
  out_o           = out_i;
  if (flg_out==1) *out = out_o;
  valid_out_o = valid_out_i;
  if (flg_valid_out==1) *valid_out = valid_out_o;
  /* END   : Register Assignment */
  goto L;
}
/* *out = $ (a_tmp + b_tmp); */
out_i      = a_tmp + b_tmp;          /* Refined */
*out          = out_o;               /* Refined */
flg_out       = 1;                   /* Added */
/* *valid_out = $ 0x0001; */
valid_out_i     = 0x0001;         /* Refined */
*valid_out           = valid_out_o; /* Refined */
flg_valid_out   = 1;              /* Added */
} else {
  /* $ */
  /* BEGIN : Register Assignment */
  valid_a_tmp_o   = valid_a_tmp_i;
  valid_a_tmp     = valid_a_tmp_o;
  out_o        = out_i;
  if (flg_out==1) *out = out_o;
  valid_out_o = valid_out_i;
  if (flg_valid_out==1) *valid_out = valid_out_o;
  /* END   : Register Assignment */
  /* *valid_out = $ 0x0000; */
  valid_out_i = 0x0000;           /* Refined */
  *valid_out      = valid_out_o;    /* Refined */
  flg_valid_out   = 1;            /* Added */
 }
}
}
```

10/531287

FIG.29

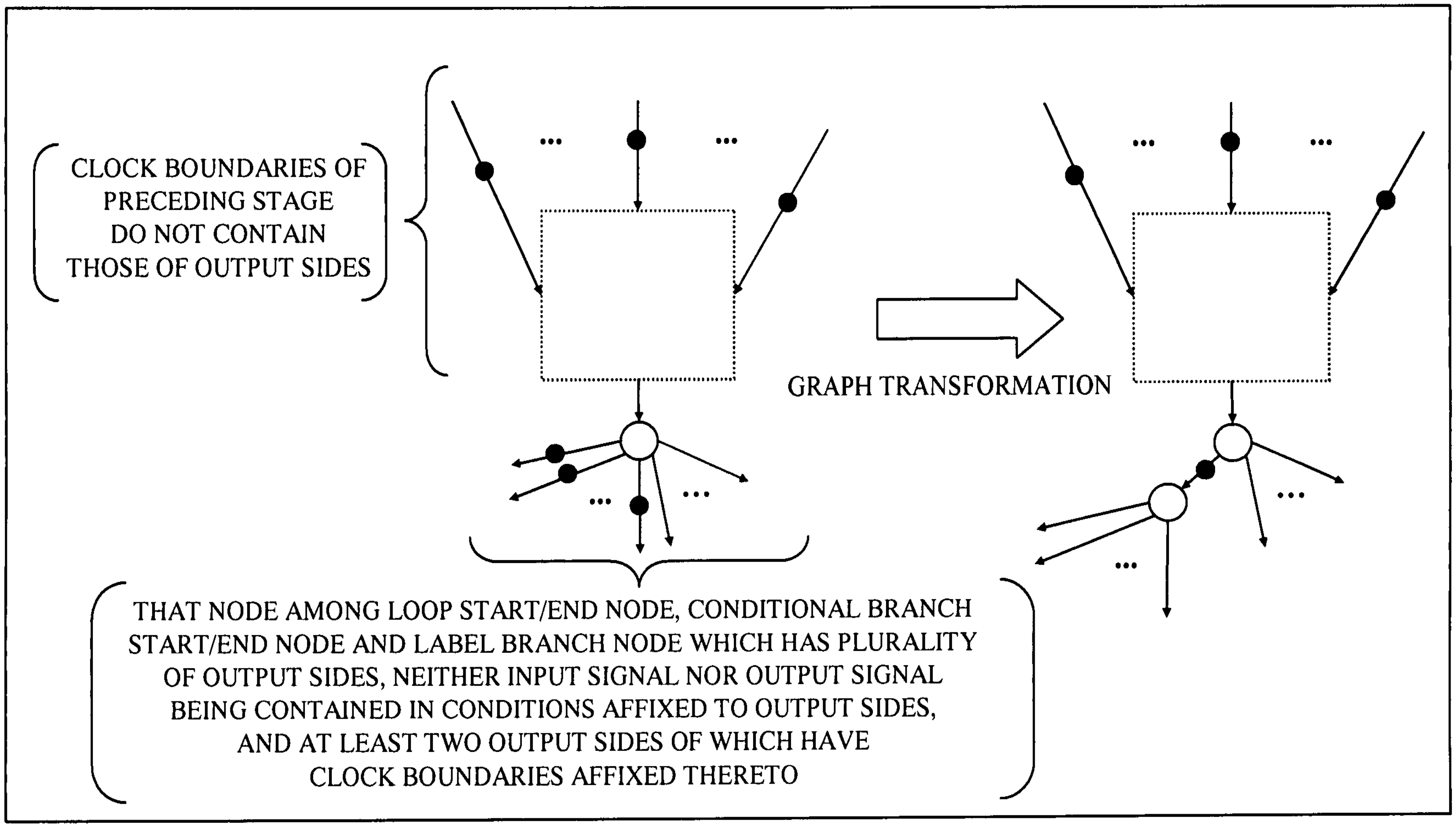
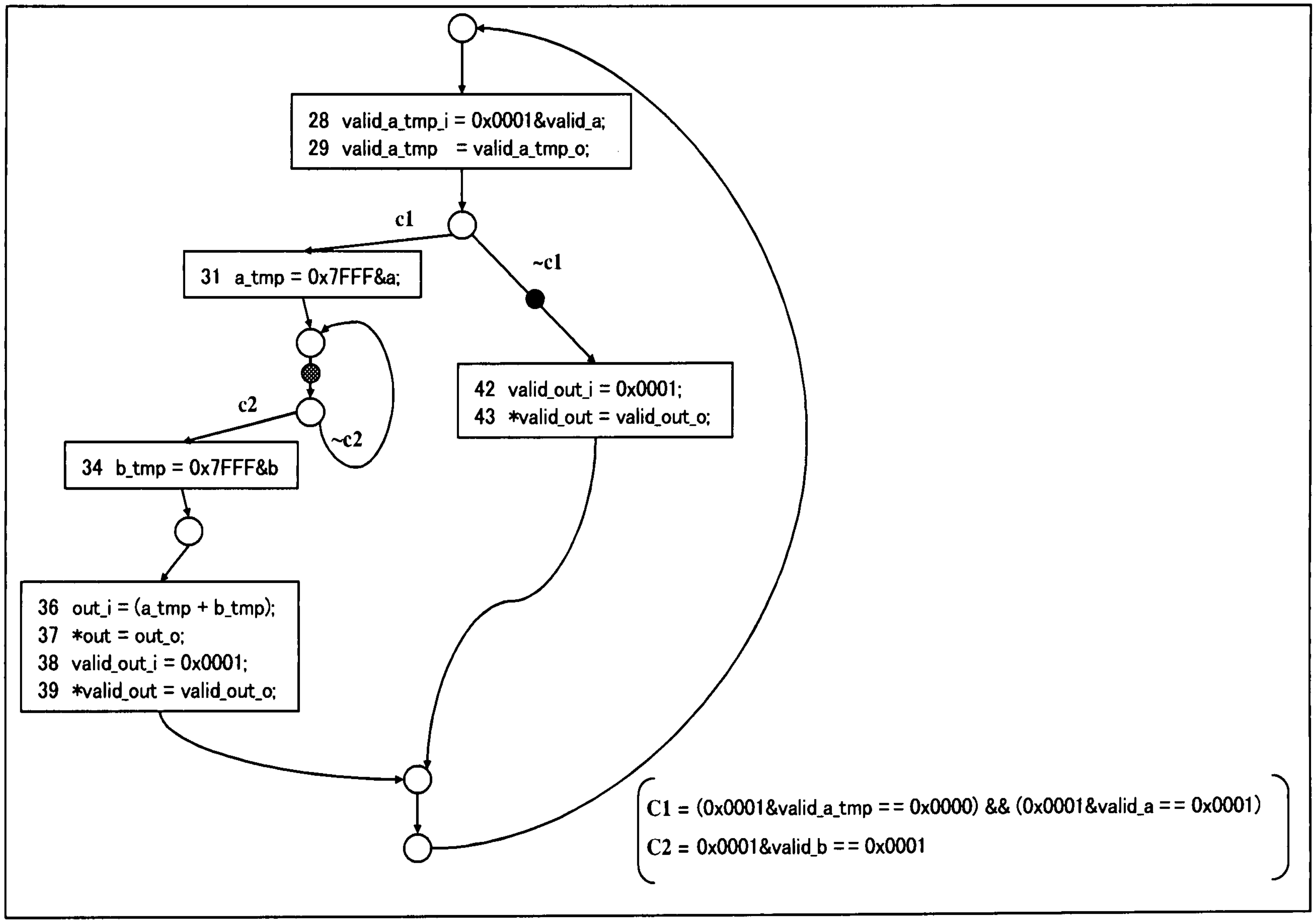

CLOCK BOUNDARIES OF PRECEDING STAGE DO NOT CONTAIN THOSE OF OUTPUT SIDES
GRAPH TRANSFORMATION
THAT NODE AMONG LOOP START/END NODE, CONDITIONAL BRANCH START/END NODE AND LABEL BRANCH NODE WHICH HAS PLURALITY OF OUTPUT SIDES, NEITHER INPUT SIGNAL NOR OUTPUT SIGNAL BEING CONTAINED IN CONDITIONS AFFIXED TO OUTPUT SIDES, AND AT LEAST TWO OUTPUT SIDES OF WHICH HAVE CLOCK BOUNDARIES AFFIXED THERETO

FIG.30



28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
~c1
31 a_tmp = 0x7FFF&a;
c2
~c2
34 b_tmp = 0x7FFF&b
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

10/531287

10/531287

FIG.31

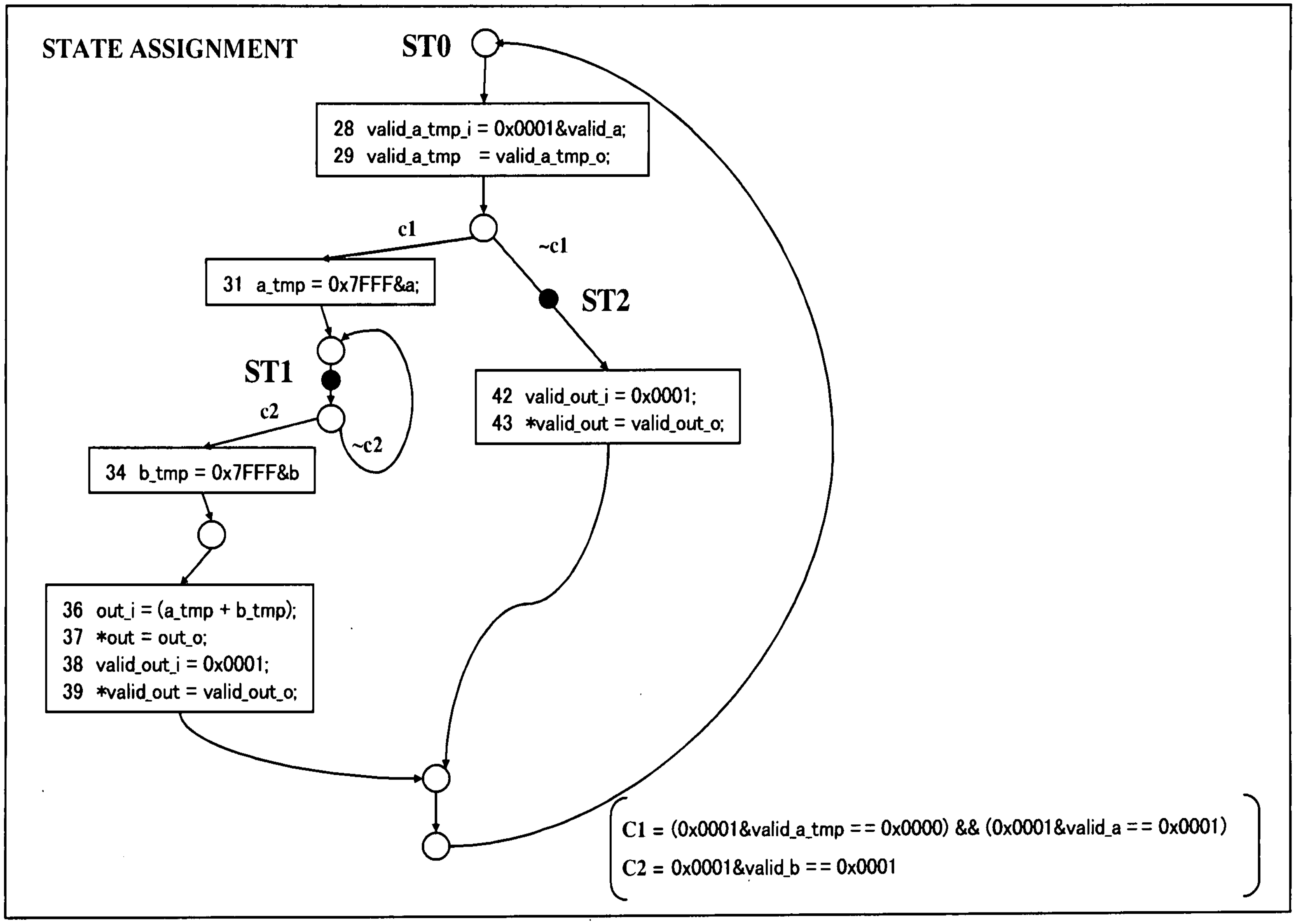
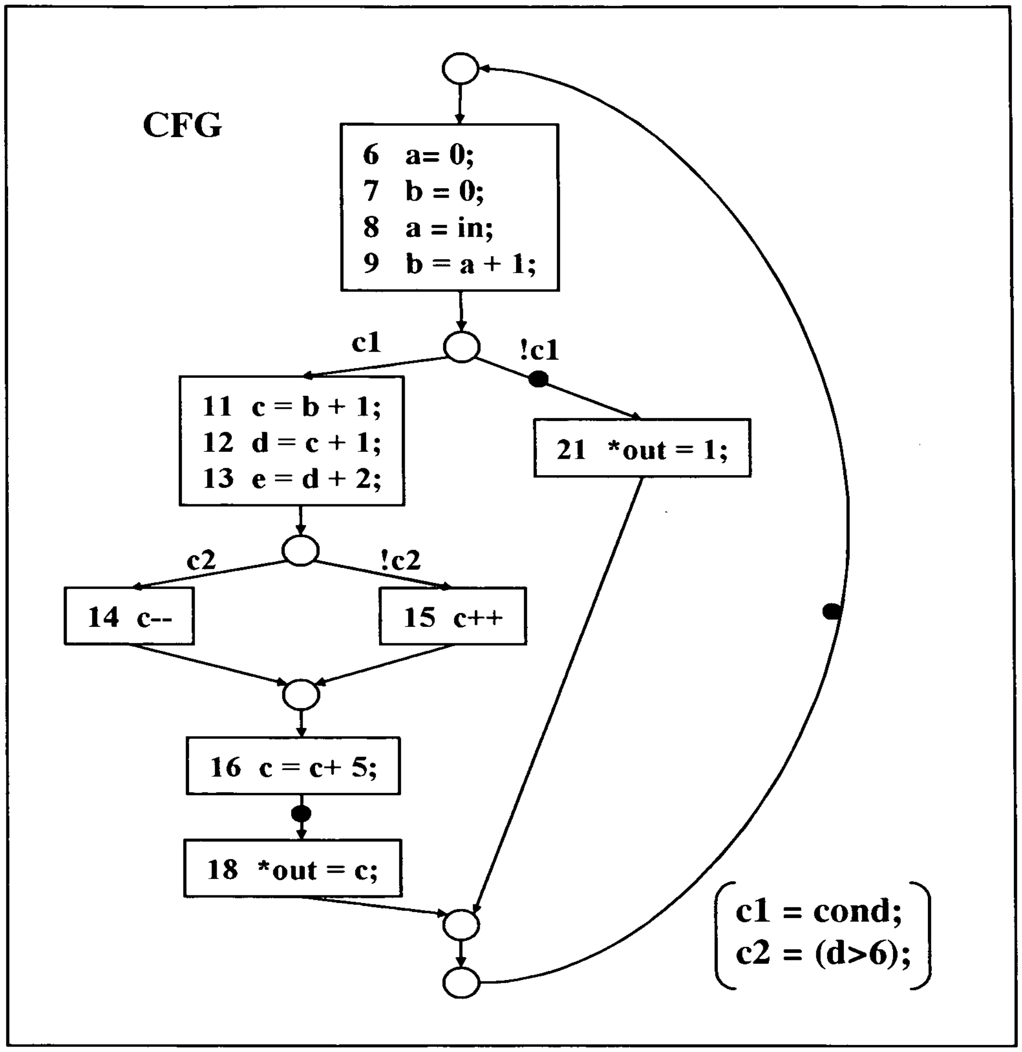

STATE ASSIGNMENT
ST0
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
~c1
31 a_tmp = 0x7FFF&a;
ST2
ST1
c2
~c2
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
34 b_tmp = 0x7FFF&b
36 out_i = (a_tmp + b_tmp);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001

FIG.32

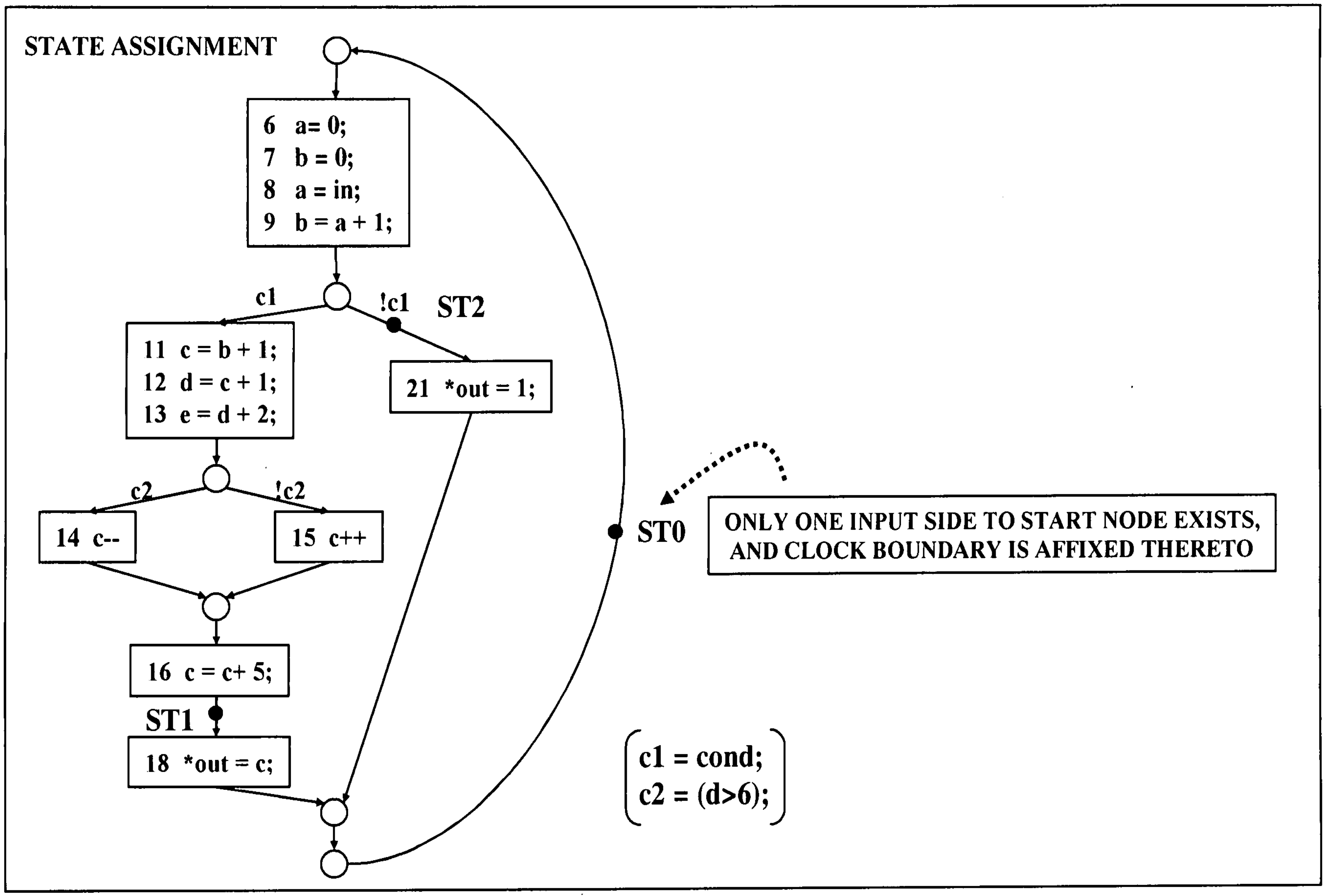
```
void foo(unsigned short in,
             unsigned short cond,
             unsigned short *out) {
  unsigned short a, b, c, d, e;
  while(1) {
    a= 0;
    b = 0;
    a = in;
    b = a + 1;
     if (cond) {
       c = b + 1;
       d = c + 1;
       e  = d + 2;
       if (d > 6) c--;
       else        c++;
       c = c + 5;
       $
       *out = c;
     } else {
       $
       *out = 1;
     }
     $
 }
```

FIG.33


CFG
6 a= 0;
7 b = 0;
8 a = in;
9 b = a + 1;
c1
!c1
11 c = b + 1;
12 d = c + 1;
13 e = d + 2;
21 *out = 1;
c2
!c2
14 c--
15 c++
16 c = c+ 5;
18 *out = c;
c1 = cond;
c2 = (d>6);

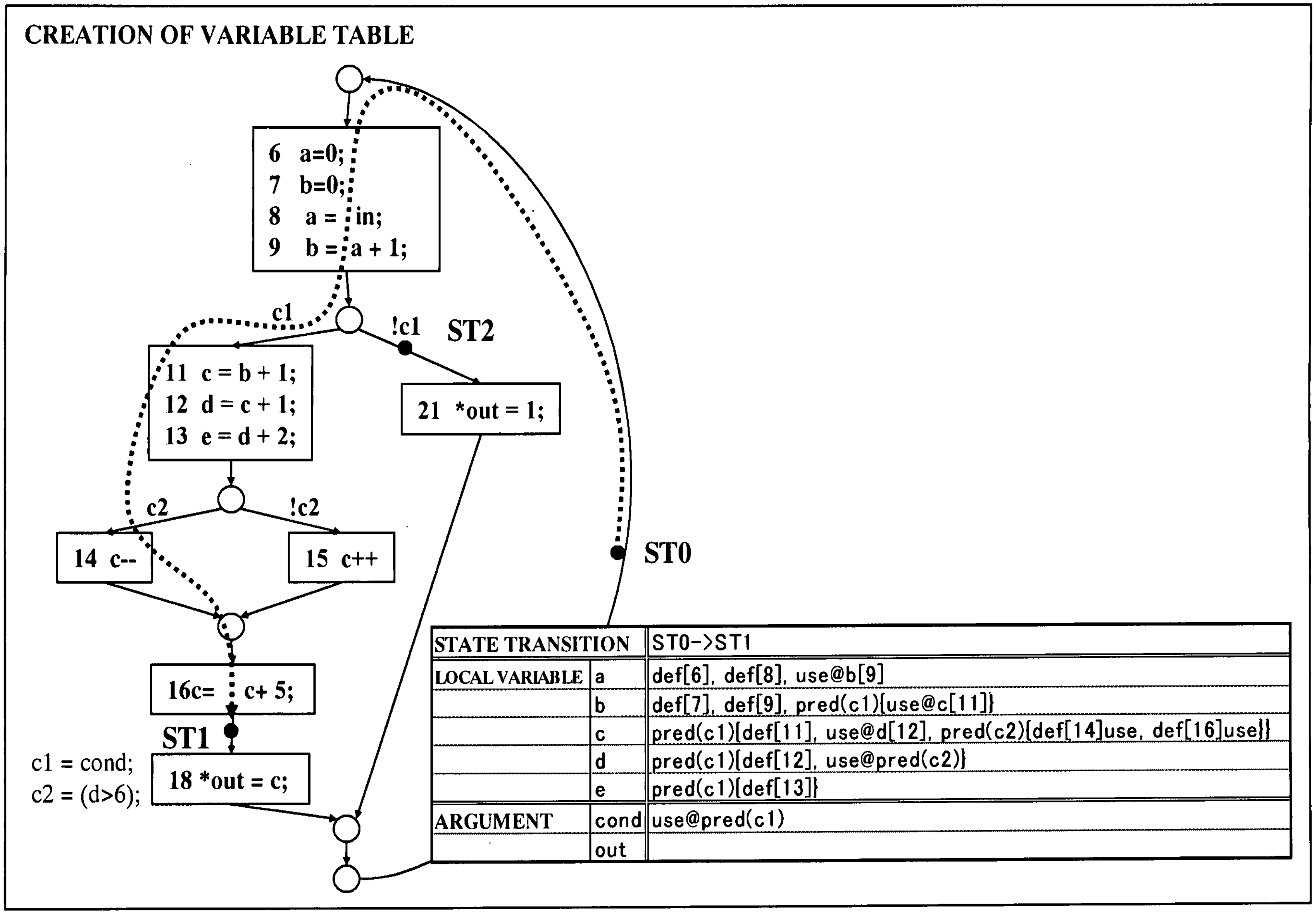
10/531287

FIG.34



10/531287

FIG.35

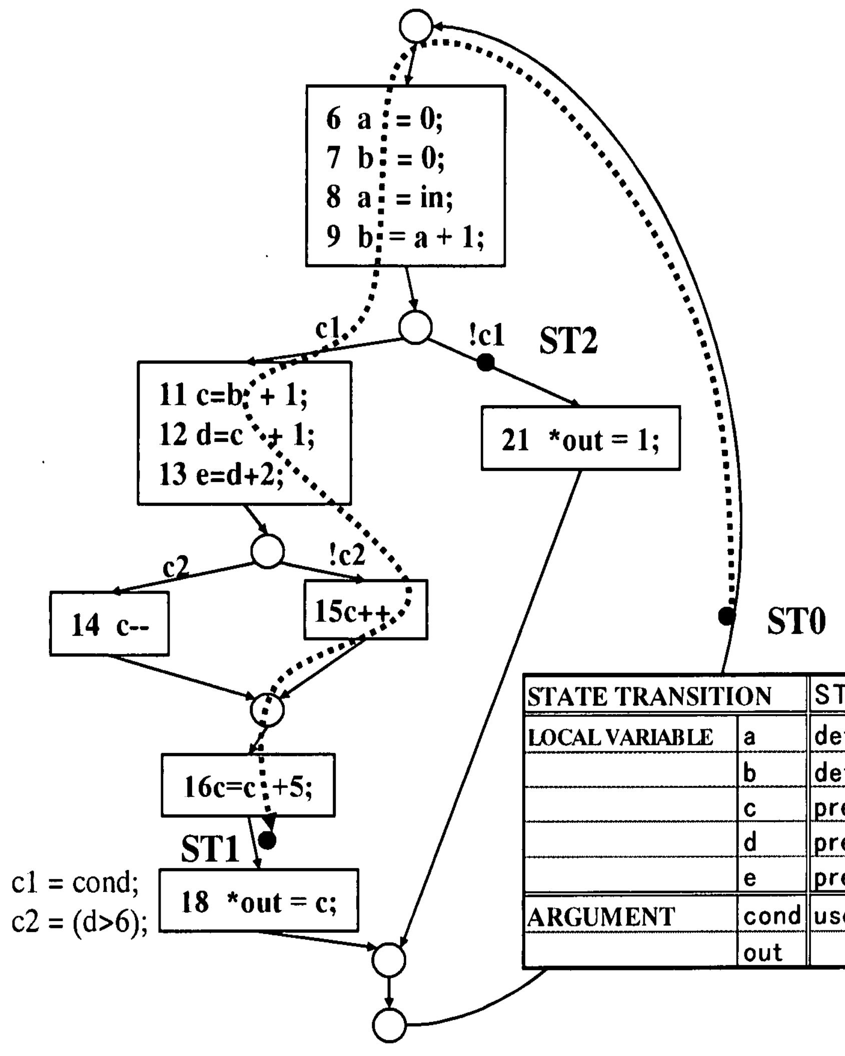


STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[6], def[8], use@b[9]
	b	def[7], def[9], pred(c1){use@c[11]}
	c	pred(c1){def[11], use@d[12], pred(c2){def[14]use, def[16]use}}
	d	pred(c1){def[12], use@pred(c2)}
	e	pred(c1){def[13]}
ARGUMENT	cond	use@pred(c1)
	out	

10/531287

FIG.36

CREATION OF VARIABLE TABLE

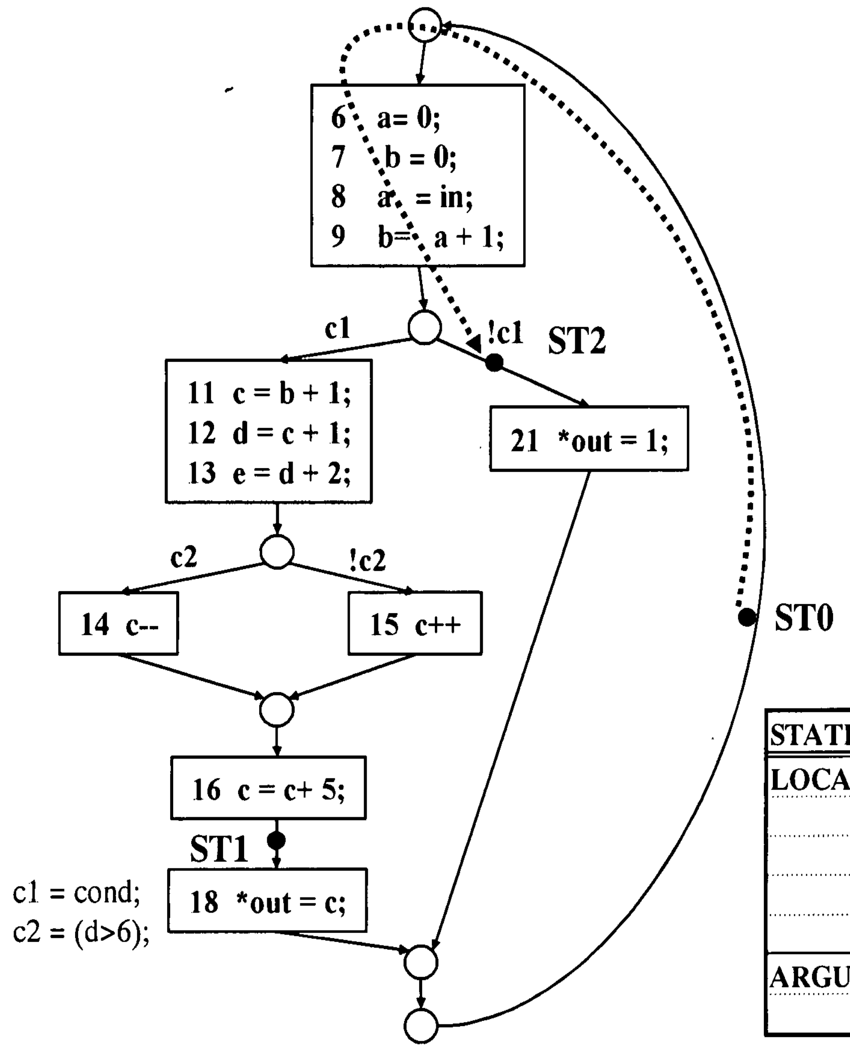


STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[6], def[8], use@b[9]
	b	def[7], def[9], pred(c1)[use@c[11]}
	c	pred(c1)[def[11], use@d[12], pred(!c2)[def[14]use, def[16]use]}
	d	pred(c1)[def[12], use@pred(!c2)}
	e	pred(c1)[def[13]}
ARGUMENT	cond	use@pred(c1)
	out	

10/531287

FIG.37

CREATION OF VARIABLE TABLE

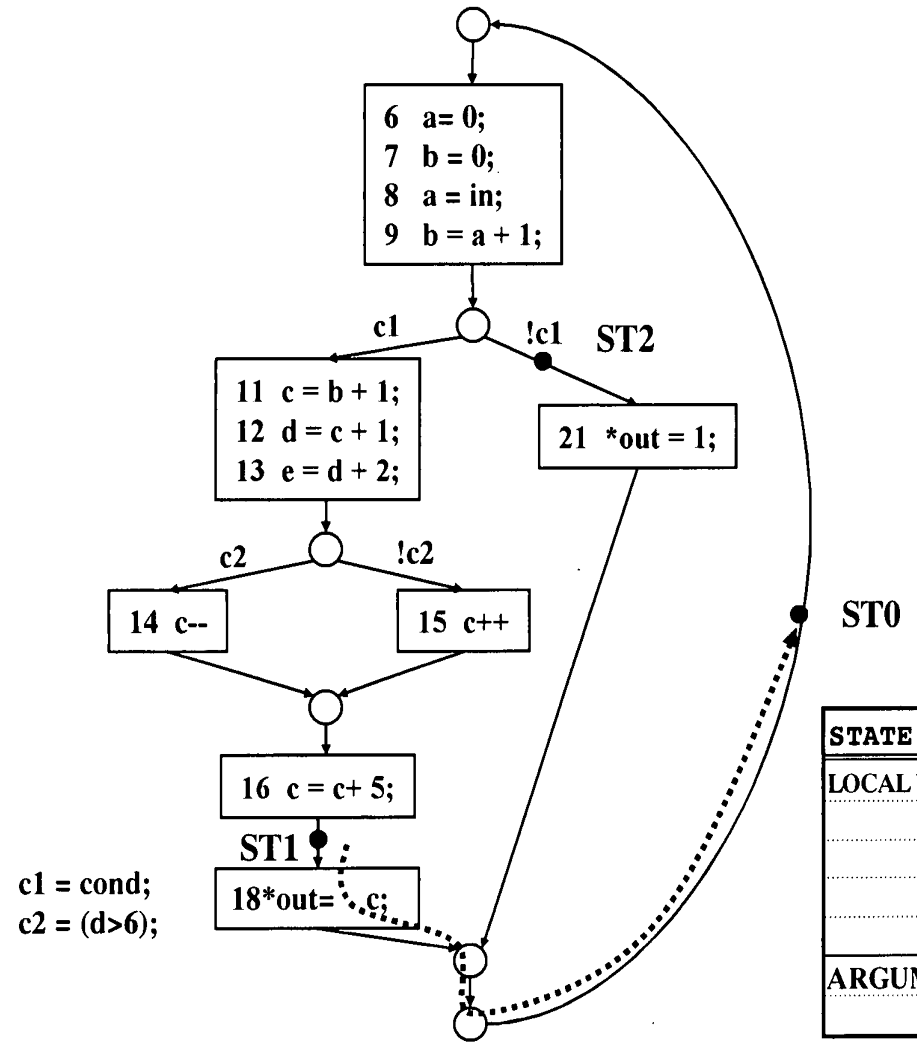


STATE TRANSITION		ST0->ST2
LOCAL VARIABLE	a	def[6], def[8], use@b[9]
	b	def[7], def[9]
	c	
	d	
	e	
ARGUMENT	cond	use@pred(!c1)
	out	

10/531287

FIG.38

CREATION OF VARIABLE TABLE

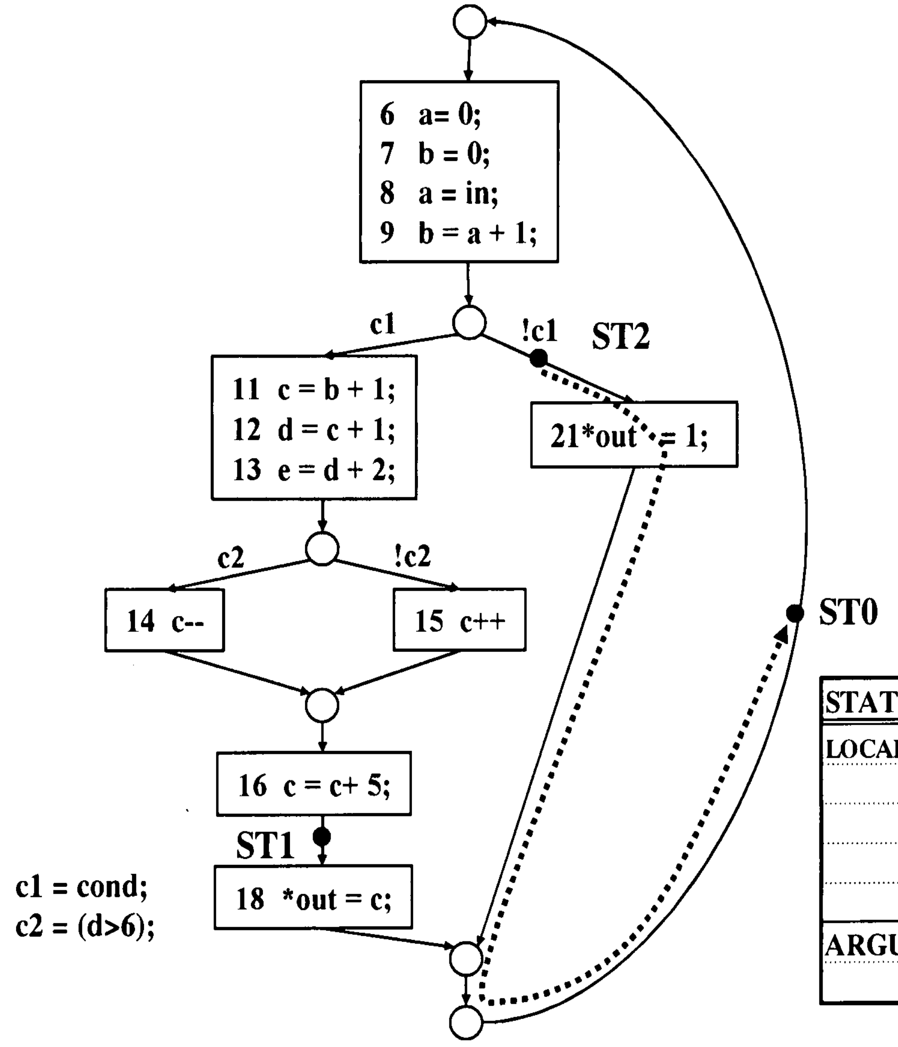


STATE TRANSITION		ST1->ST0
LOCAL VARIABLE	a	
	b	
	c	use@out[17]
	d	
	e	
ARGUMENT	cond	
	out	def[18]

10/531287

FIG.39

CREATION OF VARIABLE TABLE



STATE TRANSITION		ST2->ST0
LOCAL VARIABLE	a	
	b	
	c	
	d	
	e	
ARGUMENT	cond	
	out	def[21]

10/531287

FIG.40

VARIABLE TABLE CREATION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	a	def[6], def[8], use@b[9]	def[6], def[8], use@b[9]		
	b	def[7], def[9], pred(c1){use@c[11]}	def[7], def[9]		
	c	pred(c1){def[11], use@d[12], pred(c2){def[14]use, def[16]use}}		use@out[17]	
	d	pred(c1){def[12], use@pred(c2)}			
	e	pred(c1){def[13]}			
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out			def[18]	def[21]

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[6], def[8], use@b[9]
	b	def[7], def[9], pred(c1){use@c[11]}
	c	pred(c1){def[11], use@d[12], pred(!c2){def[14]use, def[16]use}}
	d	pred(c1){def[12], use@pred(!c2)}
	e	pred(c1){def[13]}
ARGUMENT	cond	use@pred(c1)
	out	

def[n]	: EXPRESSING THAT VARIABLE IS DEFINED AT LINE “n”
use@var[m]	: EXPRESSING THAT VARIABLE IS USED FOR ASSIGNMENT TO VARIABLE “var” AT LINE “m”
pred(cond){...}	: EXPRESSING THAT {...} IS PERFORMED IN CASE WHERE BRANCH OF CONDITION “cond” HAS HELD
def[l]use	: EXPRESSING THAT VARIABLE IS USED FOR ASSIGNMENT TO VARIABLE ITSELF AT LINE 1
use@pred(cond)	: EXPRESSING THAT VARIABLE IS USED IN CONDITION “cond”

10/531287

FIG.41

REDUNDANT STATEMENT DELETION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	a	~~def[6]~~, def[8], use@b[9]	~~def[6]~~, def[8], use@b[9]		
	b	~~def[7]~~, def[9], pred(c1){use@c[11]}	~~def[7]~~, def[9]		
	c	pred(c1){def[11], use@d[12], pred(c2){def[14]use, def[16]use}}		use@out[17]	
	d	pred(c1){def[12], use@pred(c2)}			
	e	pred(c1){~~def[13]~~}			
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out			def[18]	def[21]

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	~~def[6]~~, def[8], use@b[9]
	b	~~def[7]~~, def[9], pred(c1){use@c[11]}
	c	pred(c1){def[11], use@d[12], pred(!c2){def[14]use, def[16]use}}
	d	pred(c1){def[12], use@pred(!c2)}
	e	pred(c1){~~def[13]~~}
ARGUMENT	cond	use@pred(c1)
	out	

10/531287

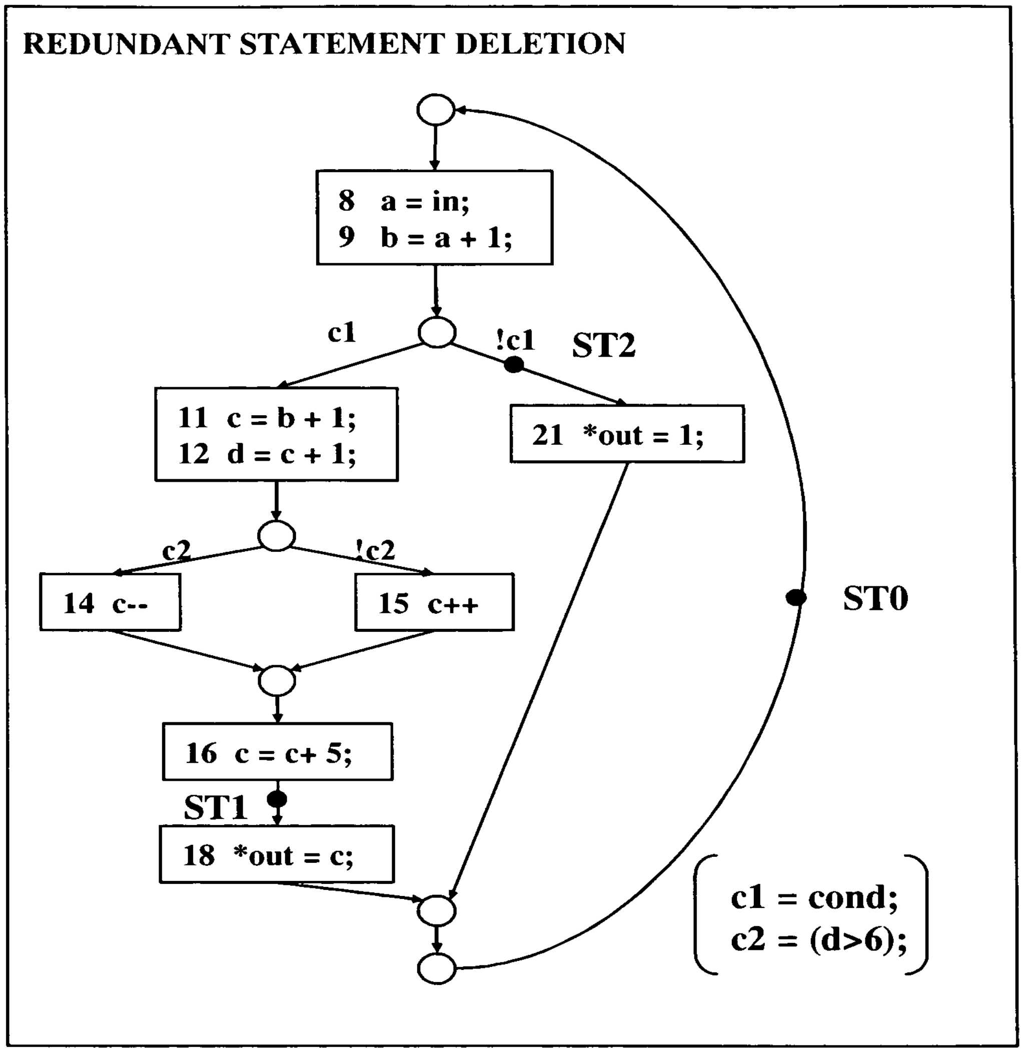
FIG. 42

REDUNDANT STATEMENT DELETION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	a	def[8], use@b[9]	def[8], use@b[9]		
	b	def[9], pred(c1){use@c[11]}	def[9]		
	c	pred(c1){def[11], use@d[12], pred(c2){def[14]use, def[16]use}}		use@out[17]	
	d	pred(c1){def[12], use@pred(c2)}			
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out			def[18]	def[21]

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	def[8], use@b[9]
	b	def[9], pred(c1){use@c[11]}
	c	pred(c1){def[11], use@d[12], pred(!c2){def[14]use, def[16]use}}
	d	pred(c1){def[12], use@pred(!c2)}
ARGUMENT	cond	use@pred(c1)
	out	

FIG. 43


REDUNDANT STATEMENT DELETION
8 a = in;
9 b = a + 1;
c1
!c1
ST2
11 c = b + 1;
12 d = c + 1;
21 *out = 1;
c2
!c2
14 c--
15 c++
ST0
16 c = c+ 5;
ST1
18 *out = c;
c1 = cond;
c2 = (d>6);

10/531287

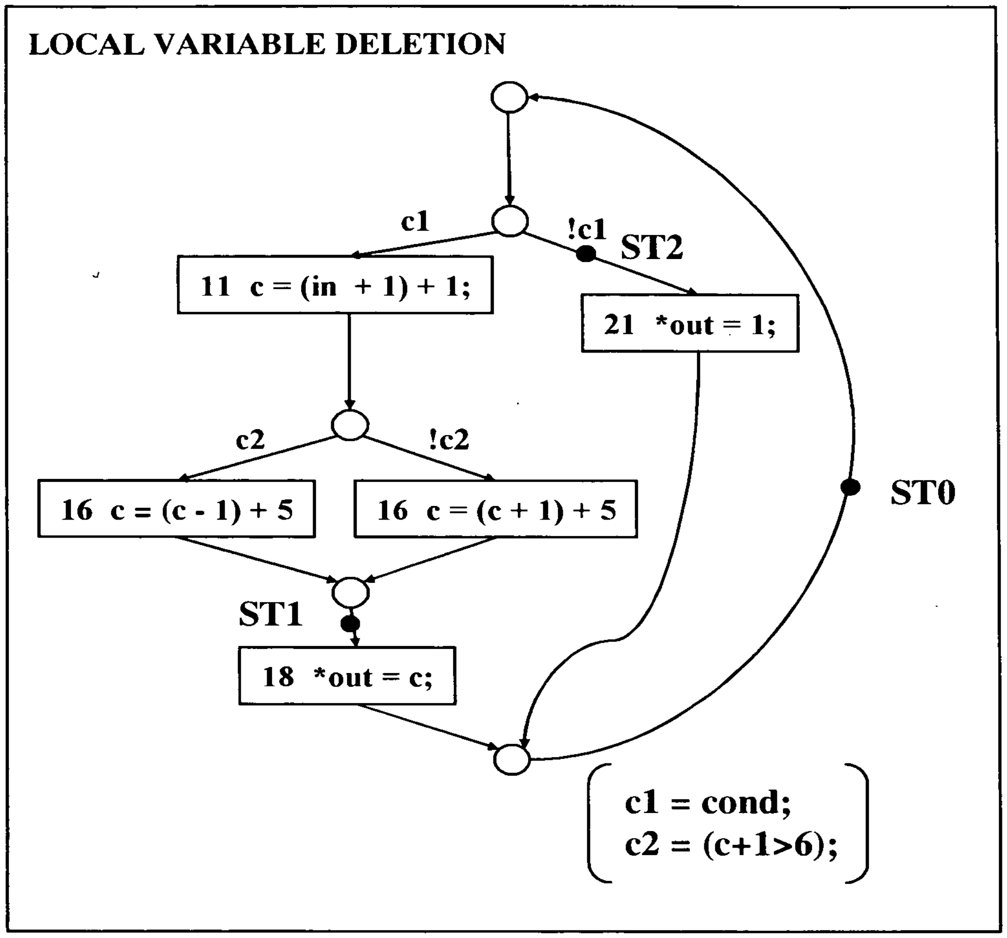
FIG.44

LOCAL VARIABLE DELETION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	a	~~def~~[8], use@b[9]	~~def~~[8], use@b[9]		
	b	~~def~~[9], pred(c1){use@c[11]}	~~def~~[9]		
	c	pred(c1){def[11], use@d[12], pred(c2){~~def[14]~~use, def[16]use}}		use@out[17]	
	d	pred(c1){~~def~~[12], use@pred(c2)}			
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out			def[18]	def[21]

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	a	~~def~~[8], use@b[9]
	b	~~def~~[9], pred(c1){use@c[11]}
	c	pred(c1){def[11], use@d[12], pred(!c2){~~def[14]~~use, def[16]use}}
	d	pred(c1){~~def~~[12], use@pred(!c2)}
ARGUMENT	cond	use@pred(c1)
	out	

FIG.45


LOCAL VARIABLE DELETION
c1
!c1
ST2
11 c = (in + 1) + 1;
21 *out = 1;
c2
!c2
16 c = (c - 1) + 5
16 c = (c + 1) + 5
ST0
ST1
18 *out = c;
c1 = cond;
c2 = (c+1>6);

10/531287

FIG. 46

AFTER UPDATING

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	c	pred(c1){def[11], pred(c2){def[16]use}}		use@out[17]	
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out			def[18]	def[21]

STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	c	pred(c1){def[11], pred(!c2){def[16]use}}
ARGUMENT	cond	use@pred(c1)
	out	

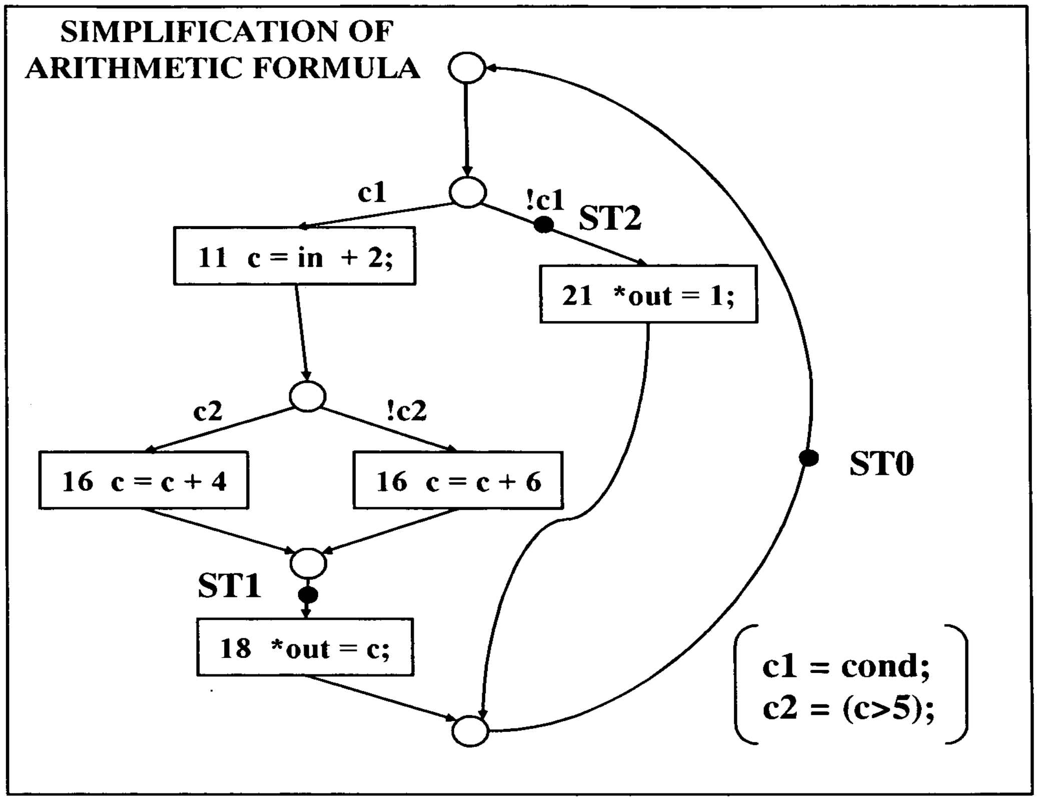
10/531287

FIG.47

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST0	ST2->ST0
LOCAL VARIABLE	c	pred(c1){def[11], pred(c2){def[16]use}}	retain	retain	retain
ARGUMENT	cond	use@pred(c1)	use@pred(!c1)		
	out	retain	retain	def[18]	def[21]

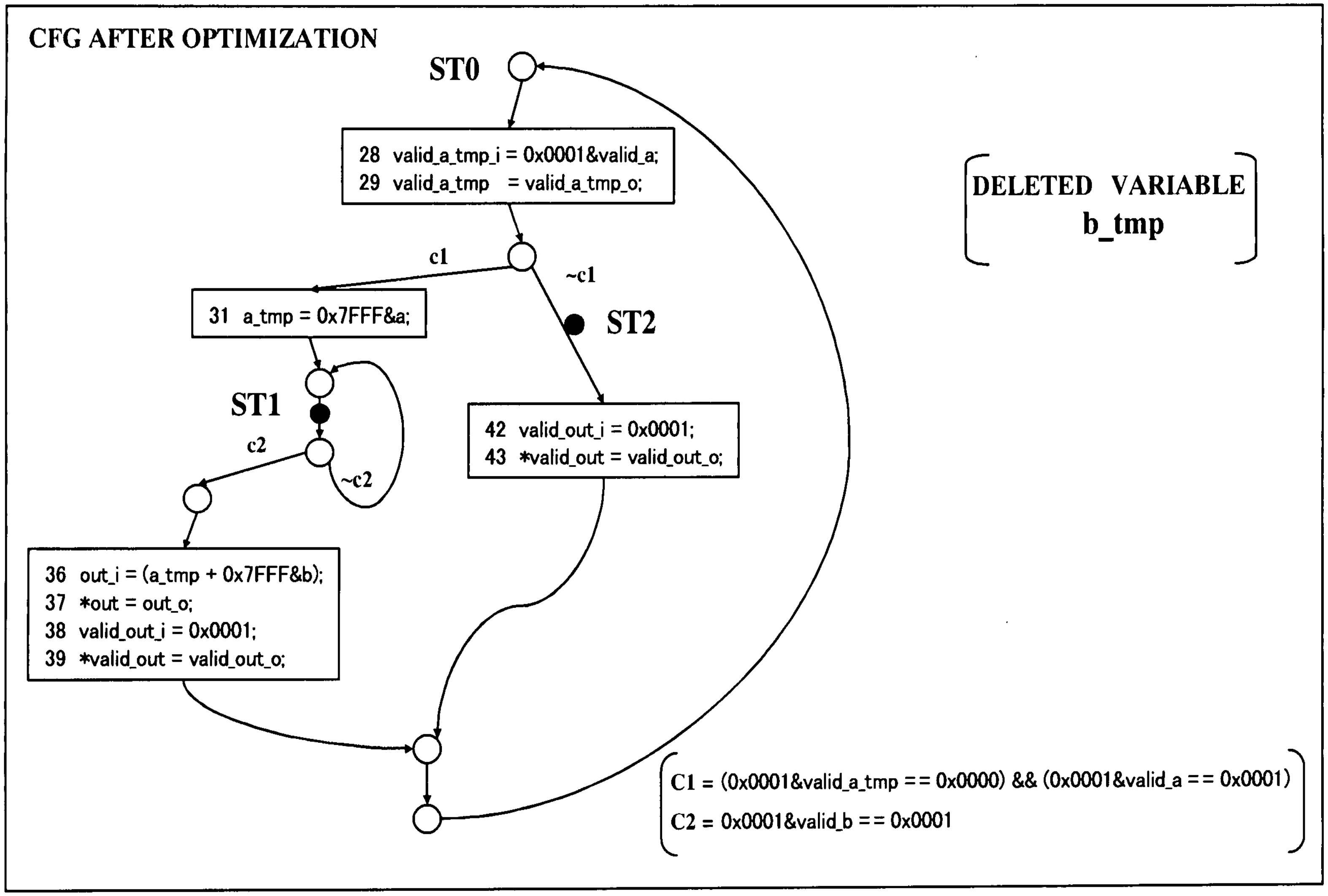
STATE TRANSITION		ST0->ST1
LOCAL VARIABLE	c	pred(c1){def[11], pred(!c2){def[16]use}}
ARGUMENT	cond	use@pred(c1)
	out	retain

FIG. 48


SIMPLIFICATION OF ARITHMETIC FORMULA
c1
!c1
ST2
11 c = in + 2;
21 *out = 1;
c2
!c2
16 c = c + 4
16 c = c + 6
ST0
ST1
18 *out = c;
c1 = cond;
c2 = (c>5);

10/531287

FIG.49



```
28  valid_a_tmp_i = 0x0001&valid_a;
29  valid_a_tmp   = valid_a_tmp_o;
```

```
31  a_tmp = 0x7FFF&a;
```

```
36  out_i = (a_tmp + 0x7FFF&b);
37  *out = out_o;
38  valid_out_i = 0x0001;
39  *valid_out = valid_out_o;
```

```
42  valid_out_i = 0x0001;
43  *valid_out = valid_out_o;
```

```
C1 = (0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)
C2 = 0x0001&valid_b == 0x0001
```

10/531287

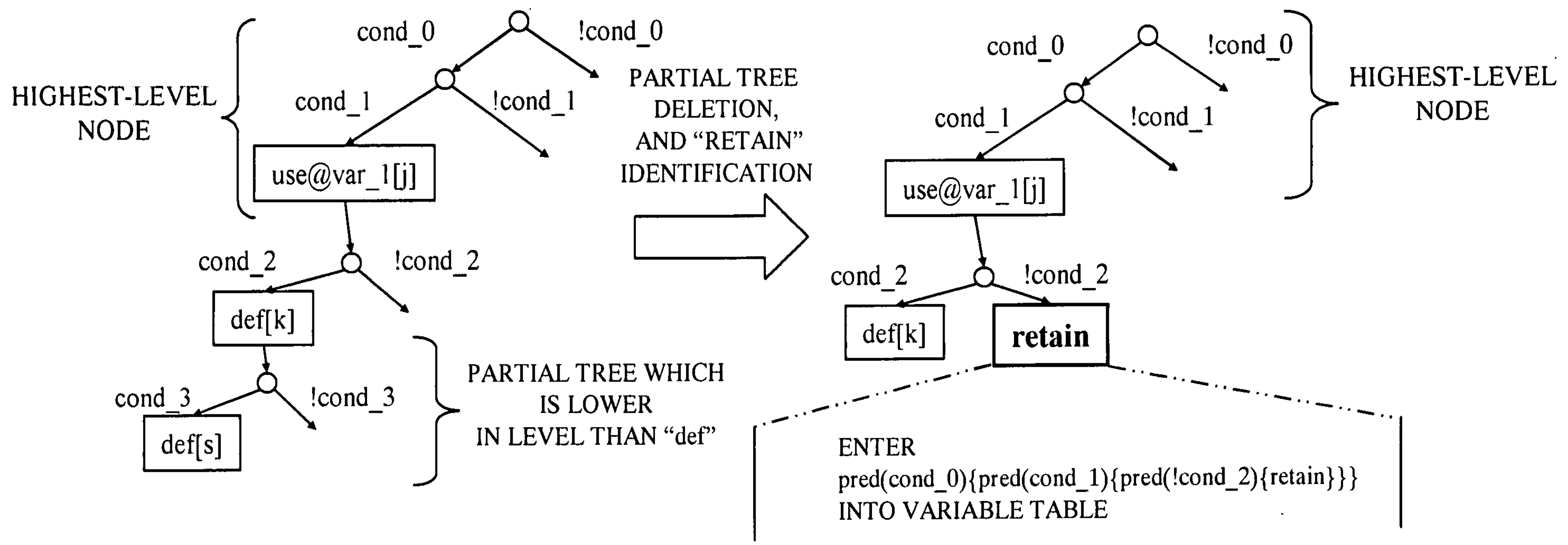
FIG.50

AFTER OPTIMIZATION

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST1
LOCAL VARIABLE	valid_a_tmp	def[29], use@pred(c1)	def[29], use@pred(!c1)	pred(c2){def[29], use@pred(c1)}
	valid_a_tmp_I	def[28]	def[28]	pred(c2){def[28]}
	a_tmp	pred(c1){def[31]}		pred(c2){pred(c1){def[31]}}
	out_I			pred(c2){def[36]}
	valid_out_i			pred(c2){def[38]}
ARGUMENT	a	pred(c1){use@a_tmp[31]}		pred(c2){pred(c1){use@a_tmp[31]}}
	b			pred(c2){use@out_i[36]}
	valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)	pred(c2){use@valid_a_tmp_i[28], use@pred(c1)}
	valid_b			use@pred(c2)
	valid_out			pred(c2){def[39]}
	out			pred(c2){def[37]}
	valid_a_tmp_o	use@valid_a_tmp[29]	use@valid_a_tmp[29]	pred(c2){use@valid_a_tmp[29]}
	valid_out_o			pred(c2){use@valid_out_o[39]}
	out_o			pred(c2){use@out[37]}
STATE TRANSITION		ST1->ST2	ST2->ST1	ST2->ST2
LOCAL VARIABLE	valid_a_tmp	pred(c2){def[29], use@pred(!c1)}	def[29], use@pred(c1)	def[29], use@pred(!c1)
	valid_a_tmp_I	pred(c2){def[28]}	def[28]	def[28]
	a_tmp		pred(c1){def[31]}	
	out_I	pred(c2){def[36]}		
	valid_out_i	pred(c2){def[38]}	def[42]	def[42]
ARGUMENT	a		pred(c1){use@a_tmp[31]}	
	b	pred(c2){use@out_i[36]}		
	valid_a	pred(c2){use@valid_a_tmp_i[28], use@pred(c1)}	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)
	valid_b	use@pred(c2)		
	valid_out	pred(c2){def[39]}	def[43]	def[43]
	out	pred(c2){def[37]}		
	valid_a_tmp_o	pred(c2){use@valid_a_tmp[29]}		
	valid_out_o	pred(c2){use@valid_out_o[39]}	use@valid_out[43]	use@valid_out[43]
	out_o	pred(c2){use@out[37]}		

10/531287

FIG.51



10/531 287

FIG.52

AFTER EXECUTION OF "RETAIN" ANALYSIS

STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST1	ST1->ST1
LOCAL VARIABLE	valid_a_tmp	def[29], use@pred(c1)	def[29], use@pred(!c1)	pred(c2){def[29], use@pred(c1)}	pred(!c2){retain}
	valid_a_tmp_I	def[28]	def[28]	pred(c2){def[28]}	pred(!c2){retain}
	a_tmp	pred(c1){def[31]}	pred(!c1){retain}	pred(c2){pred(c1){def[31]}}	pred(!c2){retain}
	out_I	retain	retain	pred(c2){def[36]}	pred(!c2){retain}
	valid_out_i	retain	retain	pred(c2){def[38]}	pred(!c2){retain}
ARGUMENT	a	pred(c1){use@a_tmp[31]}		pred(c2){pred(c1){use@a_tmp[31]}}	
	b			pred(c2){use@out_i[36]}	
	valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)	pred(c2){use@valid_a_tmp_i[28], use@pred(c1)}	
	valid_b			use@pred(c2)	use@pred(!c2)
	valid_out	retain	retain	pred(c2){def[39]}	pred(!c2){retain}
	out	retain	retain	pred(c2){def[37]}	pred(!c2){retain}
	valid_a_tmp_o	use@valid_a_tmp[29]	use@valid_a_tmp[29]	pred(c2){use@valid_a_tmp[29]}	
	valid_out_o			pred(c2){use@valid_out_o[39]}	
	out_o			pred(c2){use@out[37]}	

STATE TRANSITION		ST1->ST2	ST2->ST1	ST2->ST2
LOCAL VARIABLE	valid_a_tmp	pred(c2){def[29], use@pred(!c1)}	def[29], use@pred(c1)	def[29], use@pred(!c1)
	valid_a_tmp_I	pred(c2){def[28]}	def[28]	def[28]
	a_tmp	pred(!c1){retain}	pred(c1){def[31]}	pred(!c1){retain}
	out_I	pred(c2){def[36]}	retain	retain
	valid_out_i	pred(c2){def[38]}	def[42]	def[42]
ARGUMENT	a		pred(c1){use@a_tmp[31]}	
	b	pred(c2){use@out_i[36]}		
	valid_a	pred(c2){use@valid_a_tmp_i[28], use@pred(c1)}	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)
	valid_b	use@pred(c2)		
	valid_out	pred(c2){def[39]}	def[43]	def[43]
	out	pred(c2){def[37]}	retain	retain
	valid_a_tmp_o	pred(c2){use@valid_a_tmp[29]}		
	valid_out_o	pred(c2){use@valid_out_o[39]}	use@valid_out[43]	use@valid_out[43]
	out_o	pred(c2){use@out[37]}		

10/531287

FIG.53

INFORMATION ACQUISITION FROM VARIABLE TABLE WHICH IS "RETAIN" ANALYSIS RESULT

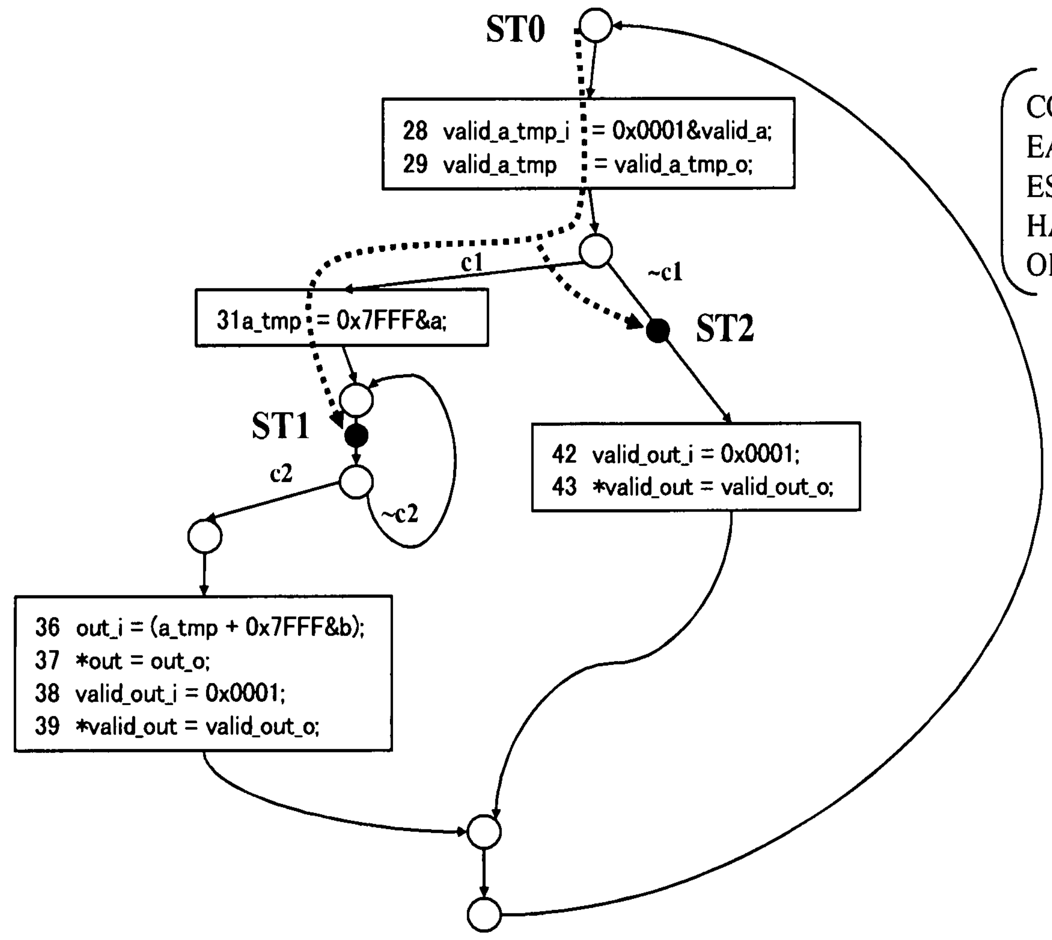
STATE TRANSITION		ST0->ST1	ST0->ST2	ST1->ST1	ST1->ST1
LOCAL VARIABLE	valid_a_tmp	def[29], use@pred(c1)	def[29], use@pred(!c1)	pred(c2){def[29], use@pred(c1)}	pred(!c2){valid_a_tmp = valid_a_tmp_o;}
	valid_a_tmp_I	def[28]	def[28]	pred(c2){def[28]}	pred(!c2){valid_a_tmp_i = valid_a_tmp_o;}
	a_tmp	pred(c1){def[31]}	pred(!c1){nxt_a_tmp = tmp;}	pred(c2){pred(c1){def[31]}}	pred(!c2){nxt_a_tmp = a_tmp;}
	out_I	out_i = out_o;	out_i = out_o;	pred(c2){def[36]}	pred(!c2){out_i = out_o;}
	valid_out_i	valid_out_i = valid_out_o;	valid_out_i = valid_out_o;	pred(c2){def[38]}	pred(!c2){valid_out_i = valid_out_o;}
ARGUMENT	a	pred(c1){use@a_tmp[31]}		pred(c2){pred(c1){use@a_tmp[31]}}	
	b			pred(c2){use@out_i[36]}	
	valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)	pred(c2){use@valid_a_tmp_i[28], use@pred(c1)}	
	valid_b			use@pred(c2)	use@pred(!c2)
	valid_out	valid_out = valid_out_o;	valid_out = valid_out_o;	pred(c2){def[39]}	pred(!c2){valid_out = valid_out_o;}
	out	out = out_o;	out = out_o;	pred(c2){def[37]}	pred(!c2){out = out_o;}
	valid_a_tmp_o	use@valid_a_tmp[29]	use@valid_a_tmp[29]	pred(c2){use@valid_a_tmp[29]}	
	valid_out_o			pred(c2){use@valid_out_o[39]}	
	out_o			pred(c2){use@out[37]}	

STATE TRANSITION		ST1->ST2	ST2->ST1	ST2->ST2
LOCAL VARIABLE	valid_a_tmp	pred(c2){def[29], use@pred(!c1)}	def[29], use@pred(c1)	def[29], use@pred(!c1)
	valid_a_tmp_I	pred(c2){def[28]}	def[28]	def[28]
	a_tmp	pred(!c1){nxt_a_tmp = a_tmp;}	pred(c1){def[31]}	pred(!c1){nxt_a_tmp = a_tmp;}
	out_I	pred(c2){def[36]}	out_i = out_o;	out_i = out_o;
	valid_out_i	pred(c2){def[38]}	def[42]	def[42]
ARGUMENT	a		pred(c1){use@a_tmp[31]}	
	b	pred(c2){use@out_i[36]}		
	valid_a	pred(c2){use@valid_a_tmp_i[28], use@pred(c1)}	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)
	valid_b	use@pred(c2)		
	valid_out	pred(c2){def[39]}	def[43]	def[43]
	out	pred(c2){def[37]}	out = out_o;	out = out_o;
	valid_a_tmp_o	pred(c2){use@valid_a_tmp[29]}		
	valid_out_o	pred(c2){use@valid_out_o[39]}	use@valid_out[43]	use@valid_out[43]
	out_o	pred(c2){use@out[37]}		

10/531287

FIG.54

START STATE : ST0

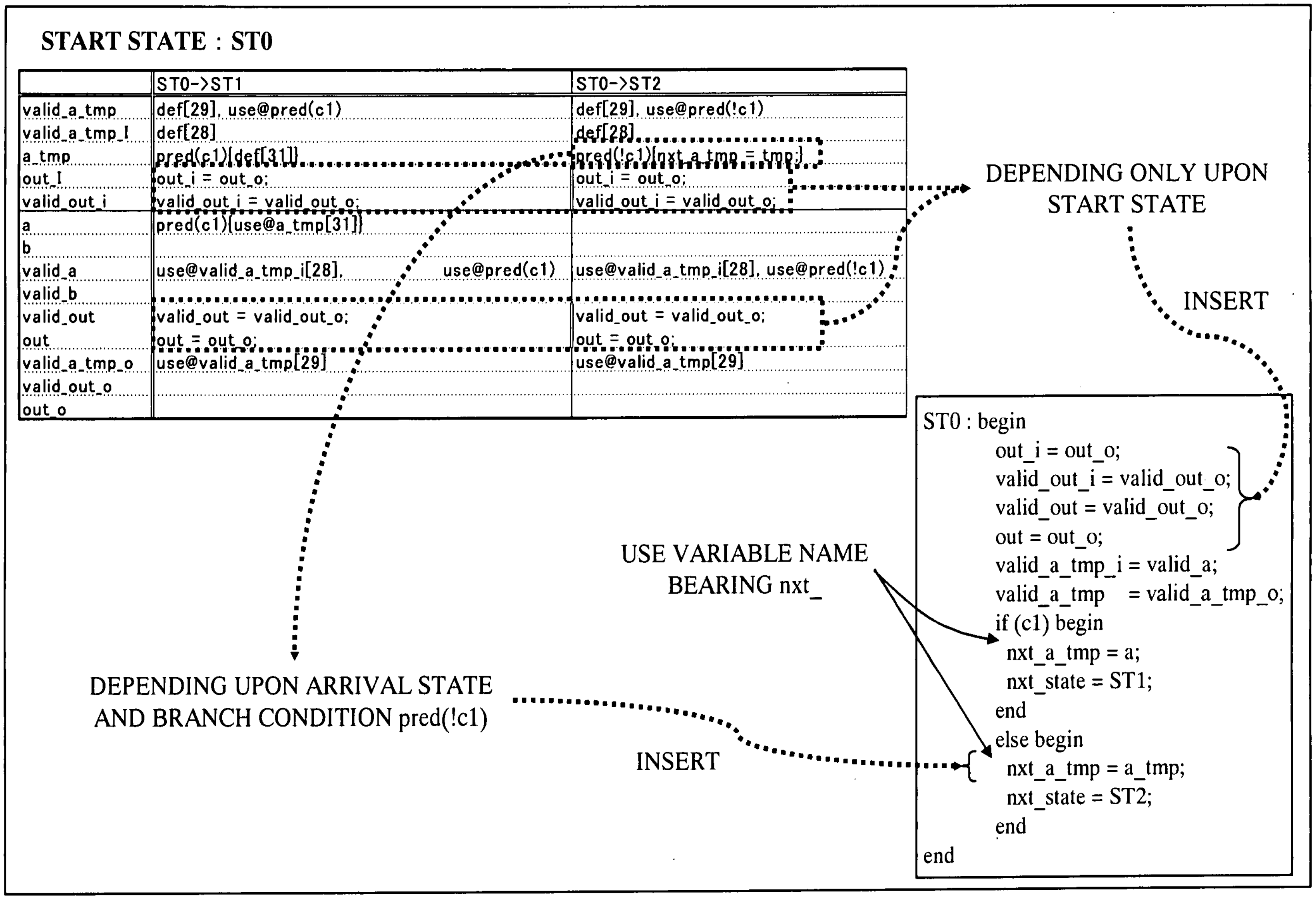

ST0
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
~c1
31a_tmp = 0x7FFF&a;
ST1
ST2
c2
~c2
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
36 out_i = (a_tmp + 0x7FFF&b);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

CODES ARE GENERATED FROM EACH STATE BY DFS. ESPECIALLY, STATE VARIABLES HAVE CODES GENERATED IN FORMS OF nxt_state = ST0, ETC.

```
ST0 : begin
          valid_a_tmp_i=  valid_a;
          valid_a_tmp    =
valid_a_tmp_o;
          if (c1) begin           } ST1への分岐
            nxt_a_tmp = a;
            nxt_state = ST1;
          end
          else begin              } ST2への分岐
            nxt_state = ST2;
          end
end
```

10/531287

FIG.55



START STATE : ST0

	ST0->ST1	ST0->ST2
valid_a_tmp	def[29], use@pred(c1)	def[29], use@pred(!c1)
valid_a_tmp_I	def[28]	def[28]
a_tmp	pred(c1)[def[31]]	pred(!c1)[nxt_a_tmp = tmp;]
out_I	out_i = out_o;	out_i = out_o;
valid_out_i	valid_out_i = valid_out_o;	valid_out_i = valid_out_o;
a	pred(c1)[use@a_tmp[31]]	
b		
valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)
valid_b		
valid_out	valid_out = valid_out_o;	valid_out = valid_out_o;
out	out = out_o;	out = out_o;
valid_a_tmp_o	use@valid_a_tmp[29]	use@valid_a_tmp[29]
valid_out_o		
out_o		

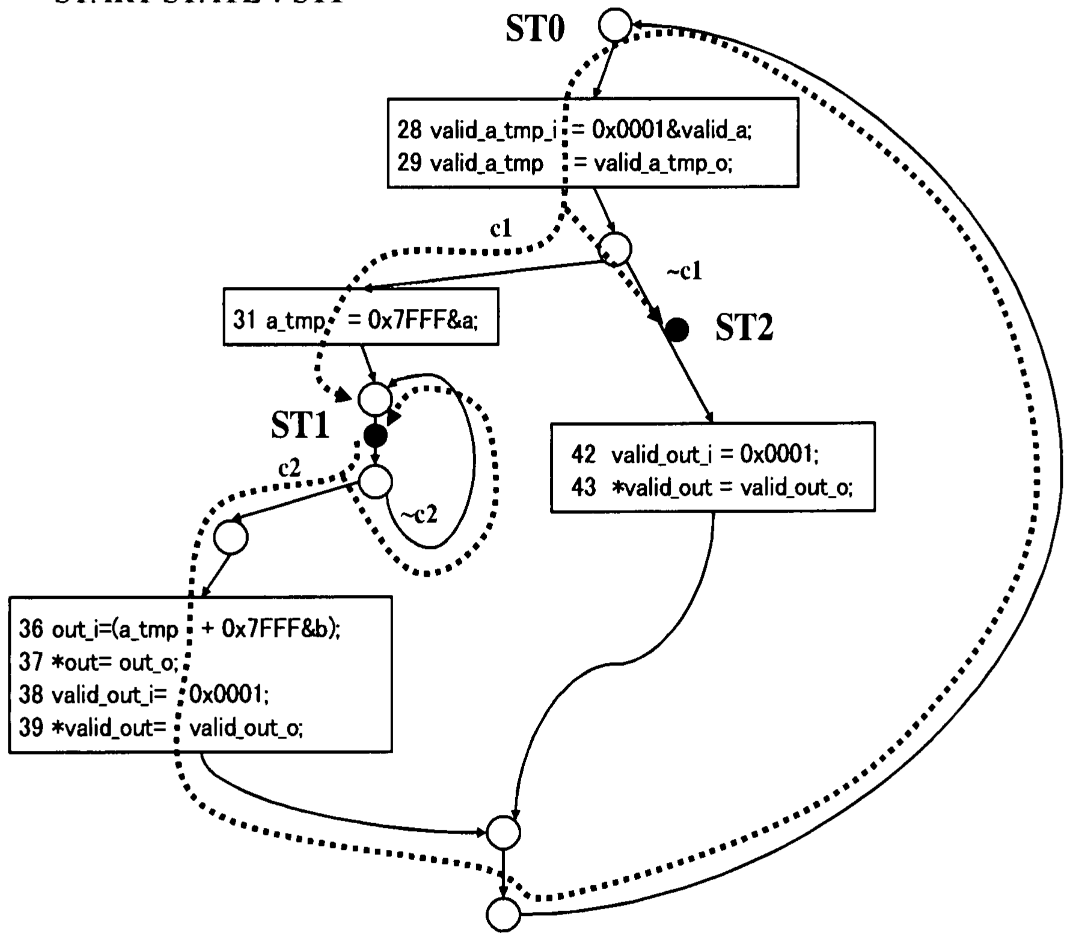
```
ST0 : begin
        out_i = out_o;
        valid_out_i = valid_out_o;
        valid_out = valid_out_o;
        out = out_o;
        valid_a_tmp_i = valid_a;
        valid_a_tmp    = valid_a_tmp_o;
        if (c1) begin
          nxt_a_tmp = a;
          nxt_state = ST1;
        end
        else begin
          nxt_a_tmp = a_tmp;
          nxt_state = ST2;
        end
end
```

DEPENDING ONLY UPON START STATE
INSERT
USE VARIABLE NAME BEARING nxt_
DEPENDING UPON ARRIVAL STATE AND BRANCH CONDITION pred(!c1)
INSERT

10/531287

FIG.56

START STATE : ST1

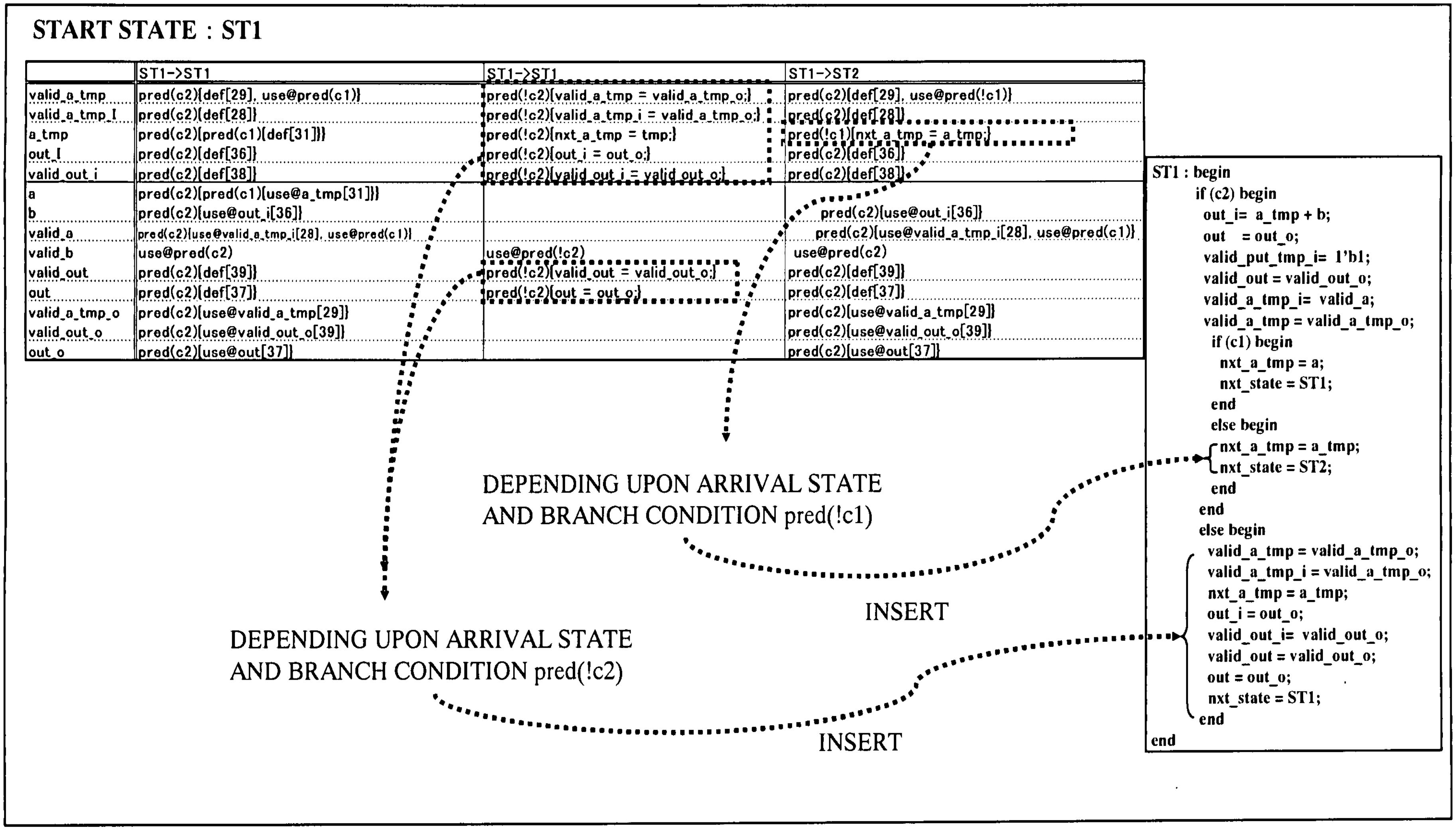


ST0
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
~c1
ST2
31 a_tmp = 0x7FFF&a;
ST1
c2
~c2
42 valid_out_i = 0x0001;
43 *valid_out = valid_out_o;
36 out_i=(a_tmp + 0x7FFF&b);
37 *out= out_o;
38 valid_out_i= 0x0001;
39 *valid_out= valid_out_o;

```
ST1 : begin
        if (c2) begin
          out_i = a_tmp + b;
          out    = out_o;
          valid_put_tmp_i = 1'b1;
          valid_out = valid_out_o;
          valid_a_tmp_i = valid_a;
          valid_a_tmp = valid_a_tmp_o;
           if (c1) begin
             nxt_a_tmp = a;
             nxt_state = ST1;
           end
           else begin
             nxt_state = ST2;
           end
         end
         else begin
           nxt_state = ST1;
         end
end
```

10/531287

FIG.57



START STATE : ST1

	ST1->ST1	ST1->ST1	ST1->ST2
valid_a_tmp	pred(c2){def[29], use@pred(c1)}	pred(!c2){valid_a_tmp = valid_a_tmp_o;}	pred(c2){def[29], use@pred(!c1)}
valid_a_tmp_I	pred(c2){def[28]}	pred(!c2){valid_a_tmp_i = valid_a_tmp_o;}	pred(c2){def[28]}
a_tmp	pred(c2){pred(c1){def[31]}}	pred(!c2){nxt_a_tmp = tmp;}	pred(!c1){nxt_a_tmp = a_tmp;}
out_I	pred(c2){def[36]}	pred(!c2){out_i = out_o;}	pred(c2){def[36]}
valid_out_i	pred(c2){def[38]}	pred(!c2){valid_out_i = valid_out_o;}	pred(c2){def[38]}
a	pred(c2){pred(c1){use@a_tmp[31]}}		
b	pred(c2){use@out_i[36]}		pred(c2){use@out_i[36]}
valid_a	pred(c2){use@valid_a_tmp_i[28], use@pred(c1)}		pred(c2){use@valid_a_tmp_i[28], use@pred(c1)}
valid_b	use@pred(c2)	use@pred(!c2)	use@pred(c2)
valid_out	pred(c2){def[39]}	pred(!c2){valid_out = valid_out_o;}	pred(c2){def[39]}
out	pred(c2){def[37]}	pred(!c2){out = out_o;}	pred(c2){def[37]}
valid_a_tmp_o	pred(c2){use@valid_a_tmp[29]}		pred(c2){use@valid_a_tmp[29]}
valid_out_o	pred(c2){use@valid_out_o[39]}		pred(c2){use@valid_out_o[39]}
out_o	pred(c2){use@out[37]}		pred(c2){use@out[37]}

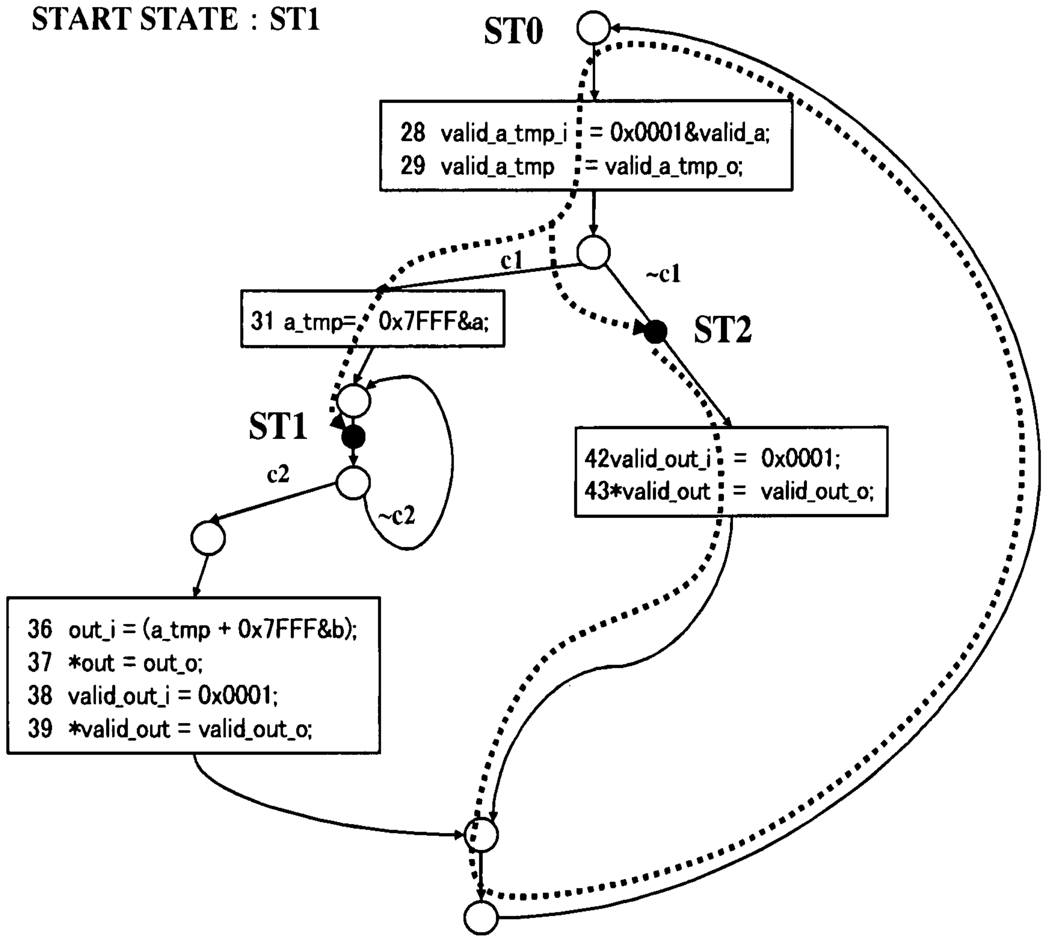
```
ST1 : begin
    if (c2) begin
     out_i=  a_tmp + b;
     out    = out_o;
     valid_put_tmp_i=  1'b1;
     valid_out = valid_out_o;
     valid_a_tmp_i=  valid_a;
     valid_a_tmp = valid_a_tmp_o;
      if (c1) begin
        nxt_a_tmp = a;
        nxt_state = ST1;
      end
      else begin
        nxt_a_tmp = a_tmp;
        nxt_state = ST2;
      end
    end
    else begin
     valid_a_tmp = valid_a_tmp_o;
     valid_a_tmp_i = valid_a_tmp_o;
     nxt_a_tmp = a_tmp;
     out_i = out_o;
     valid_out_i=  valid_out_o;
     valid_out = valid_out_o;
     out = out_o;
     nxt_state = ST1;
    end
end
```

DEPENDING UPON ARRIVAL STATE AND BRANCH CONDITION pred(!c1)
INSERT
DEPENDING UPON ARRIVAL STATE AND BRANCH CONDITION pred(!c2)
INSERT

10/531287

FIG.58

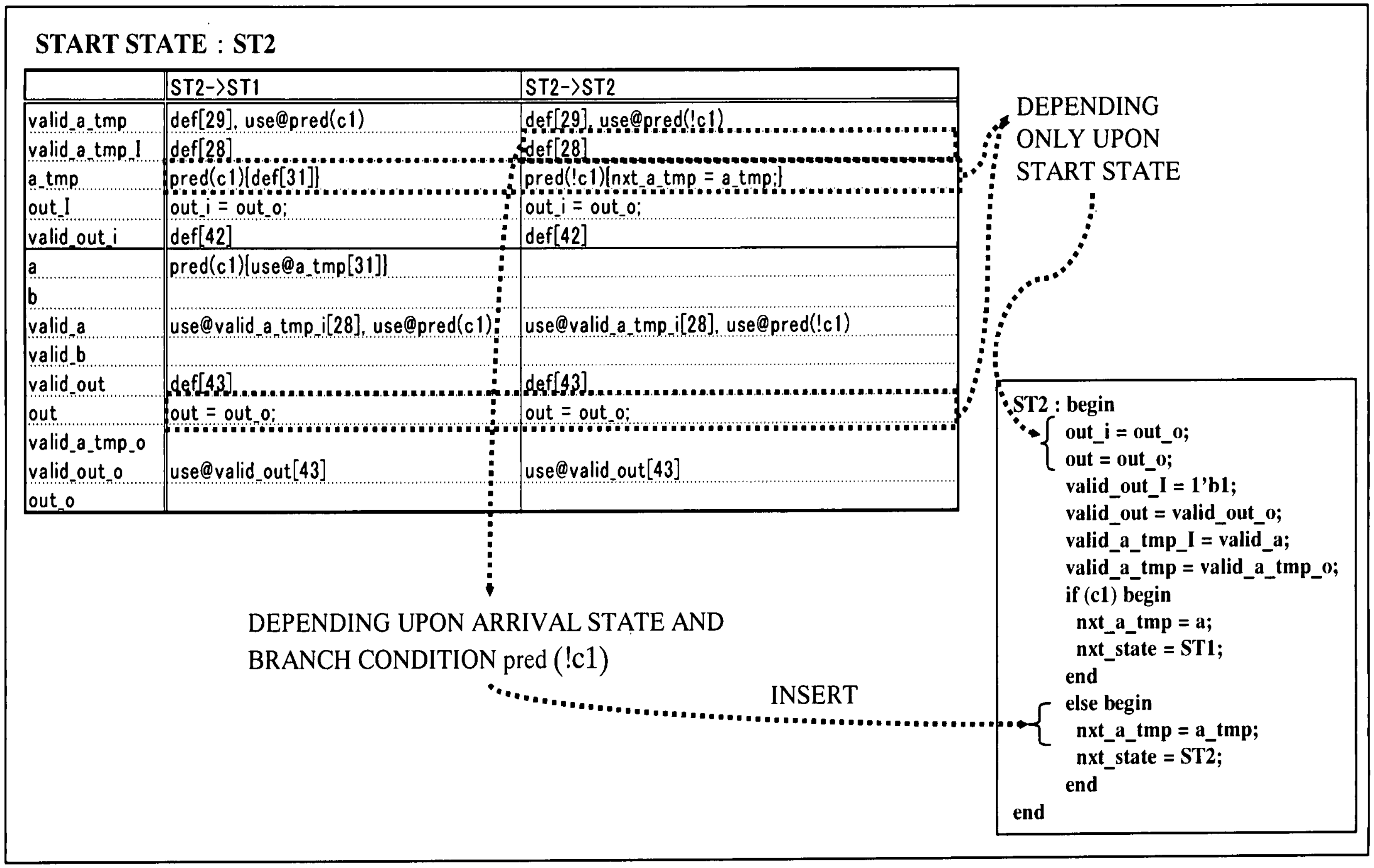
START STATE : ST1


ST0
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp = valid_a_tmp_o;
c1
~c1
ST2
31 a_tmp= 0x7FFF&a;
ST1
c2
~c2
42valid_out_i = 0x0001;
43*valid_out = valid_out_o;
36 out_i = (a_tmp + 0x7FFF&b);
37 *out = out_o;
38 valid_out_i = 0x0001;
39 *valid_out = valid_out_o;

```
ST2 : begin
        valid_out_I = 1'b1;
        valid_out = valid_out_o;
        valid_a_tmp_I = valid_a;
        valid_a_tmp = valid_a_tmp_o;
        if (c1) begin
          nxt_a_tmp = a;
          nxt_state = ST1;
        end
        else begin
          nxt_state = ST2;
        end
end
```

10/531287

FIG.59



START STATE : ST2

	ST2->ST1	ST2->ST2
valid_a_tmp	def[29], use@pred(c1)	def[29], use@pred(!c1)
valid_a_tmp_I	def[28]	def[28]
a_tmp	pred(c1)[def[31]]	pred(!c1)[nxt_a_tmp = a_tmp;]
out_I	out_i = out_o;	out_i = out_o;
valid_out_i	def[42]	def[42]
a	pred(c1)[use@a_tmp[31]]	
b		
valid_a	use@valid_a_tmp_i[28], use@pred(c1)	use@valid_a_tmp_i[28], use@pred(!c1)
valid_b		
valid_out	def[43]	def[43]
out	out = out_o;	out = out_o;
valid_a_tmp_o		
valid_out_o	use@valid_out[43]	use@valid_out[43]
out_o		

DEPENDING ONLY UPON START STATE
DEPENDING UPON ARRIVAL STATE AND BRANCH CONDITION pred (!c1)
INSERT

```
ST2 : begin
    out_i = out_o;
    out = out_o;
    valid_out_I = 1'b1;
    valid_out = valid_out_o;
    valid_a_tmp_I = valid_a;
    valid_a_tmp = valid_a_tmp_o;
    if (c1) begin
      nxt_a_tmp = a;
      nxt_state = ST1;
    end
    else begin
      nxt_a_tmp = a_tmp;
      nxt_state = ST2;
    end
end
```

FIG.60

```
module PipeLine(clk, reset_n,
                valid_a, valid_b, a, b,
                out, valid_out);
  // System clock and reset
input clk;
input reset_n;
  // PipeLine input signals
input valid_a;
input valid_b;
input [14:0] a;
input [14:0] b;
  // PipeLine output signals
output valid_out;
reg valid_out;
output [15:0] out;
reg [15:0] out;
// PipeLine internal signals
reg valid_a_tmp;
reg valid_a_tmp_i;
reg valid_a_tmp_o;
reg [14:0] a_tmp;
reg [14:0] nxt_a_tmp;
reg valid_out_i;
reg valid_out_o;
reg [15:0] out_i;
reg [15:0] out_o;
// State registers
reg [1:0] state, nxt_state;
parameter ST0=2'b00,
          ST1=2'b01,
          ST2=2'b10;
// Blanch conditions
wire c1;
wire c2;
assign c1 = !valid_a_tmp&&valid_a;
assign c2 = valid_b;
```

10/531287

FIG.61

```
// Regsiter assignment statement
always @ (posedge clk or negedge reset_n) begin
  if (!reset_n) begin
    valid_a_tmp_o <= 1'b0;
    out_o         <= 17'b00000000000000000;
  end
  else begin
    valid_a_tmp_o <= valid_a_tmp_i;
    out_o         <= out_i;
  end
end
// State registers and temporal registers
always @ (posedge clk or negedge reset_n) begin
  if (!reset_n) begin
    state         <= ST0;
    a_tmp         <= 16'b0;
  end
  else begin
    state         <= nxt_state;
    a_tmp         <= nxt_a_tmp;
  end
end
```

```
// Mealy finite state machine
always @ (state or c1 or c2 or
                valid_a_tmp_i or valid_a_tmp_o or
                valid_a_tmp or a_tmp or
                valid_out_i or valid_out_o or
                out_i or out_o) begin
    case(state[1:0])
    ST0 : begin
             valid_a_tmp_i   = valid_a;
             valid_a_tmp = valid_a_tmp_o;
             valid_out_i = valid_out_o;
             valid_out       = valid_out_o;
             out_i           = out_o;
             out             = out_o;
             if (c1) begin
               nxt_a_tmp = a;
               nxt_state = ST1;
             end
             else begin
               nxt_a_tmp = a_tmp;
               nxt_state = ST2;
             end
    end
```

10/531287

FIG.62

```
ST1 : begin
      if (c2) begin
        out_i         = a_tmp + b;
        out           = out_o;
       valid_out_i = 1'b1;
       valid_out      = valid_out_o;
        valid_a_tmp_i   = valid_a;
        valid_a_tmp = valid_a_tmp_o;
        if (c1) begin
          nxt_a_tmp = a;
          nxt_state = ST1;
        end
        else begin
          nxt_a_tmp = a_tmp;
          nxt_state = ST2;
        end
      end
      else begin
        nxt_state       = ST1;
        valid_a_tmp_i   = valid_a_tmp_o;
        valid_a_tmp = valid_a_tmp_o;
        nxt_a_tmp       = a_tmp;
       valid_out_i = valid_out_o;
      valid_out       = valid_out_o;
      out_i           = out_o;
      out             = out_o;
      end
 end
ST2 : begin
        valid_a_tmp_i   = valid_a;
        valid_a_tmp = valid_a_tmp_o;
       valid_out_i = 1'b0;
       valid_out       = valid_out_o;
       out_i           = out_o;
       out             = out_o;
        if (c1) begin
          nxt_a_tmp = a;
          nxt_state = ST1;
        end
        else begin
          nxt_a_tmp = a_tmp;
          nxt_state = ST2;
        end
end
default : begin
  nxt_state       = ST0;
  valid_a_tmp_i   = valid_a_tmp_o;
  valid_a_tmp = 1'b0;
  nxt_a_tmp       = 15'b0;
  valid_out_i = valid_out_o;
  valid_out       = 1'b0;
  out_i           = out_o;
  out             = out_o;
end
endcase
end
endmodule
```